#4

PATENT APPLICATION

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of

Ruben GONZALES

Appln. No.: 09/937,096 Group Art Unit: Not yet assigned

Confirmation No.: Not yet Assigned Examiner: Not yet assigned

Filed: September 20, 2001

For: AN OBJECT ORIENTED VIDEO SYSTEM

SUBMISSION OF PRIORITY DOCUMENTS

Commissioner for Patents
Washington, D.C. 20231

Sir:

Submitted herewith are two certified copies of the priority documents on which claims to priority were made under 35 U.S.C. § 119. The Examiner is respectfully requested to acknowledge receipt of said priority documents.

Respectfully submitted,

[illegible]

Richard C. Turner
Registration No. 29,710

SUGHRUE MION, PLLC
2100 Pennsylvania Avenue, N.W.
Washington, D.C. 20037-3213
Telephone: (202) 293-7060
Facsimile: (202) 293-7860

Enclosures: Australia PQ3603
Australia PQ8661

Date: December 19, 2001

PCT/AU00/01296

AUSTRALIA

4

Patent Office
Canberra

REC'D 29 NOV 2000
WIPO PCT

I, JONNE YABSLEY, ACTING TEAM LEADER EXAMINATION SUPPORT & SALES hereby certify that annexed is a true copy of the Provisional specification in connection with Application No. PQ 8661 for a patent by ACTIVESKY, INC. filed on 07 July 2000.

COMMONWEALTH OF AUSTRALIA
AUSTRALIA
PATENT OFFICE

WITNESS my hand this
Twenty-first day of November 2000

[illegible]

JONNE YABSLEY
ACTING TEAM LEADER
EXAMINATION SUPPORT & SALES

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH RULE 17.1(a) OR (b)

P/00/003
Regulation 3.2

ACTIVESKY, INC.

AUSTRALIA

Patents Act 1990

PROVISIONAL SPECIFICATION

for the invention entitled:

"An Object Oriented Video System"

The invention is described in the following statement:

AN OBJECT ORIENTED VIDEO SYSTEM

Field of the Invention

The present invention relates to an video encoding and processing method, and in particular, but not exclusively, to a video encoding system which supports the coexistence of multiple arbitrarily-shaped video objects in a video scene and permits individual animations and interactive behaviours to be defined for each object, and permits dynamic media composition by encoding object oriented controls into video streams that can be decoded by remote thin client systems. The thin client systems may be executed on a standard computer or mobile computer devices, such as personal digital assistants (PDAs), smart wireless phones, hand-held computers and wearable computing devices. These may include wireless transmission of the encoded video streams.

Background

Recent technology improvements have resulted in the introduction of personal mobile computing devices which are just beginning to include full wireless communication technologies. The global uptake of wireless mobile telephones has been significant, but is ready for substantial growth. There have, however, not been any video technology solutions which have provided the video quality, frame rate or low power consumption required for potential new and innovative mobile video processes. Due to the limited processing power of mobile devices, there are currently no suitable mobile video solutions for processes utilising personal computing devices such as mobile video conferencing, ultra-thin wireless network client computing, broadcast wireless mobile video, mobile video promotions or wireless video surveillance.

Computer based video conferencing currently uses standard computer workstations or PCs connected through a network consisting of a physical cable connection and network computer communication protocol layers. An example of this is a videoconference between two PCs over the Internet, with physically connected cables end to end, using the TCP/IP network communication protocols. This kind of video conferencing requires a physical connection to the Internet and also requires the use of large computer based video monitoring equipment. It provides for a videoconference between fixed locations, which additionally constrains the participants to a specific time for the conference to ensure that both parties will be at the appropriate locations simultaneously.

Broadcast of wireless digital textual information for personal handheld computers has only recently become feasible with advances in new and innovative wireless technologies and handheld computing devices. Handheld computing devices and mobile telephones are able to have wireless connections to wide area networks that can provide textual information to the user device. There is currently no real-time broadcast of video to wireless handheld computing devices. One important market issue for broadcast media in any form is the question of advertising and how it is to be supported. Effective advertising should be specifically targeted to both users and geographic locations.

Current video broadcast systems are unable to embed targeted advertising because of the considerable processing requirements needed to insert the advertising material into the video data streams in real time during transmission. The alternate method of pre-compositing video prior to transmission is too tedious to be performed on a regular basis. Additionally, once the advertising is embedded into the video stream, the user is unable to interact with the advertising in any manner, which reduces the effectiveness of the advertising. Significantly, more effective advertising can be achieved though interactive means.

Most video codecs exhibit poor performance with cartoons or animated content; however, there is more cartoon and animated content being produced for the Internet than video. Hence there is a need for a codec which enables efficient encoding of animations and cartoons as well as video.

Commercial and domestic security-based video surveillance systems have to date been achieved using closed circuit monitoring systems with video monitoring achieved in a central location, requiring the full-time attention of a dedicated surveillance guard. Video monitoring of multiple locations can only be achieved at the central control centre using dedicated monitoring system equipment. Security guards have no access to video from monitored locations whilst on patrol.

Network-based computing using thin client workstations involves minimal software processing on the client workstation, with the majority of software processing occurring on a server computer. Thin client computing reduces the cost of computer management due to the centralisation of information and operating software configuration. Client workstations are physically wired through standard local area networks such as 10 Base T Ethernet to the server computer. Client workstations run a minimal operating system enabling communication to the backend server computer and information display on the client video monitoring equipment. Existing systems, however, are constrained. They are typically limited to specific applications or vendor software. For example, current thin clients are unable to simultaneously service a video being displayed and a spreadsheet application.

To directly promote product in the market, sales representatives can use video demonstrations to illustrate product usage and benefits. Currently, for the mobile sales representative this involves the use of cumbersome dedicated video display equipment, which can be taken to customer locations for product demonstrations. There are no mobile handheld video display solutions available, which provide real-time video for product and market promotional purposes.

Video brochures have often been used for marketing and advertising. However, their effectiveness has always been limited because video is classically a passive medium. The effectiveness of video brochures would be dramatically improved if they could be made interactive. If this interactivity could be provided intrinsically within a codec, this would open the door to video-based E-Commerce applications. The current definition for interactive video consists of a player that is able to decompress a normal compressed video into a viewing window and overlay some separately defined buttons and invisible "hot regions" where a user's mouse click will invoke some separately predefined actions. In this typical approach, the video is stored as a separate entity from the controls, and the nature of interaction is extremely limited, since there is no integration between the video content and the external controls that are applied.

The provision of wireless access compatibilities to PDAs permits electronic books to be freed from their storage limitations by enabling real-time wireless streaming of audio-visual content to PDAs. Many corporate training applications require audiovisual information to be available wirelessly in portable devices. The nature of audiovisual training materials requires that they be interactive and provide for non-linear navigation of large amounts of stored content. This cannot be provided with the current state of the art.

Summary of the Invention

The present invention provides a video encoding method, including:

encoding video data with object control data as a video object; and

placing a plurality of said video object in a data stream.

Advantageously, the method may further include placing a plurality of said data stream in a scene packet.

Preferably a plurality of scene packets are combined to represent a video data file. System control and user control data may be included with the scene packets. First directory data may be included with the scene packets in the file, and second directory data may be included in a scene packet, and a third directory data may be included in a data stream.

The video data may include data representing video frames, audio frames, text and/or graphics.

The present invention also provides a video encoding method, including:

quantising colour data in a video stream based on a reduced representation of colours;

generating encoded video data representing said quantised colours and transparent regions; and

generating encoded audio data and object control data for transmission with said encoded video data.

Preferably the method further includes:

generating motion vectors representing colour changes in a video frame of said stream;

generating encoded text object and vector graphic object data for transmission with said encoded video data; and

generating encoded data for configuring customisable decompression transformations

Preferably the method supports real-time dynamic media composition of video and other media forms; and in particular includes dynamically compositing video content being displayed in real-time based on user interaction with video objects represented by said encoded data.

The method may provide voice commands to be used to control the video display.

The invention also provides server software to perform the real-time dynamic media composition process, dynamic bandwidth management and error control for live wireless streaming of video to wireless clients.

The encoded data may enable real-time client based colour quantisation for true colour video codecs.

The present invention also provides an apparatus and method for (i) interactive streaming of video and animation over public, local loop and privately owned wireless networks, and/or (ii) electronic interactive brochureware.

Advantageously said object control data may represent parameters for (i) rendering video and graphics objects, for (ii) defining the interactive behaviour of said objects, for (iii) creating hyperlinks to and from said objects, for (iv) defining animation paths for said objects and/or for (v) defining dynamic media composition parameters. The rendering parameters may represent object transparency, scale, volume, position and rotation. Animation paths may affect any of the rendering parameters. The hyperlinks may support nonlinear video navigation and may have other video files, individual scenes within a file, and other object streams within a scene as targets. The interactive behaviour data may include the pausing of play and looping play, returning user information back to the server, defining conditional execution of rendering actions or object behaviours, activating or deactivating object animations, defining menus, and simple forms that can register user selections. The dynamic media composition parameters may indicate how or what audio-visual objects are to be used for composing the scene being viewing based on user interaction or presets.

The encoded data stream preferably consists of three basic data forms; compressed data types such as audio and video data objects, parameters for defining/configuring the customisable decompression transformation, and object control data. Advantageously, the encoded video, audio, format definition and control data may be transmitted in respective packets for respective decoding.

The present invention also provides a video encoding method, including at least one of the following steps:

(i) selecting a reduced set of colours for each frame;

(ii) reconciling colours from frame to frame;

(iii) executing motion compensation;

(iv) determining update areas of a frame based on a perceptual colour difference measure;

(v) including further colour data to enhance image quality;

(vi) encoding a video data stream for each video object;

(vii) including in each video object animation and rendering controls and dynamic media composition controls.

The present invention also provides a video decoding method for decoding the encoded data.

The present invention also provides a dynamic colour space encoding method to permit further colour quantisation information to be sent to the client to enable real-time client based colour reduction.

The present invention also provides a method of including targeted user and/or local video advertising.

The present invention also includes executing an ultrathin client, which may be wireless, and which is able to provide access to remote servers.

The present invention also provides a method for multivideo conferencing.

The present invention also provides a method for creating object-oriented interactive video.

The present invention also provides a method for creating video menus and simple forms for registering user selections.

The present invention also provides a method for dynamic media composition.

The present invention also provides a method for creating video based ecommerce applications.

The present invention also provides a method for permitting users to customise and forward electronic greeting cards and post cards to mobile smart phones.

The present invention also provides a method for error correction for wireless streaming of multimedia data.

The present invention also provides systems for executing any one of the above methods, respectively.

The present invention also provides server software for permitting users to a method for error correction for wireless streaming of video data.

The present invention also provides a computer program for executing the steps of any one of the above methods, respectively.

The present invention also provides a video on demand system. The present invention also provides a video security system. The present invention also provides an interactive mobile video system.

The present invention also provides a method of processing spoken voice commands to control the video display.

Preferably the above method and systems are based on one of the above video encoding methods.

The present invention also provides a software program including code for controlling object oriented video and/or audio. Advantageously, the code may include IAVML instructions, why may be based on XML.

The invention also provides a wireless streaming video and animation system that may include:

(i) A portable monitor device and first wireless communication means.

(ii) A storage module such as a local server for storing compressed digital video and computer animations and incorporating communication means to download new content from other remote servers for either delayed or real-time viewing. Server software enables the user to browse and select what digital video to view from the library of all currently available videos in the memory store and to optionally perform dynamic media composition.

(iii) At least one interface module incorporating a second wireless communication means for transmission of transmittable data from the interface module to the portable monitor device, the portable monitor device incorporating means for receiving said transmittable data, converting the transmittable data to video images displaying the video images. Software permits the user to communicate with the server software to interactively browse and select which video to view.

(iv) Client-server software that manages the library of locally stored, or indexed videos and permits the user to exercise control of the playback and streaming of the video to the portable device and manages the function of the dynamic media composition process.

Preferably the portable monitor device is a personal digital assistant or mobile phone or similar hand-held processing unit. The storage module and the interface module may be housed together.

The invention also provides a method of providing wireless streaming of video and animation including at least one of the steps of:

(a) Downloading and storing compressed video and animation data from a remote server over a wide area network for later transmission;
(b) Permitting a user to browse and select what digital video data to view from the library of video data stored on the local server.
(c) transmitting the data to a portable monitor device; and
(d) processing the data to display the image on the portable monitor device.

The invention also provides a method of providing an interactive video brochure including at least one of the steps of:
(a) Creating a video brochure by specifying (i) the various scenes in the brochure and the various video objects that may occur within each scene; (ii) specifying the preset and user selectable scene navigational controls and the individual composition rules for each scene (iii) specifying rendering parameters on media objects such as transparency, location and animation paths etc, (iv) specifying controls on media objects to create forms to collect user feedback (v) integrating all of the compressed media streams and object control information into the composite data stream;
(b) processing the composite data stream and interpreting the object control information to display each scene.
(c) Processing user input to execute any relevant object controls, such as navigation through the brochure, activating animations etc, registering and user selections and other user input,
(d) Storing the user selections and user input for later uploading to the provider of the video brochures network server when a network connection becomes available;
(e) at the remote network server, receiving uploads of user selections from interactive video brochures and processing the information to integrate it into a customer/client database for later use.

The invention also provides a method of creating and sending video greeting cards to mobile devices including at least one of the steps of:

(a) providing a method to permit a customer to create the video greeting card by (i) selecting a template video scene or animation from a library, (ii) customising the template by adding user supplied text or audio objects or selecting video objects from a library to be inserted as actors in the scene.

(b) obtaining from the customer (i) identification details (iii) preferred delivery method (ii) payment details (iii) the intended recipient's mobile device number.

(c) queuing the greeting card depending on the nominated delivery method until either bandwidth becomes available or off peak transport can be obtained, polling the recipient's device to see if it is capable of processing the greeting card and if so, it is forwarding to the nominated mobile device.

Preferably when the greeting card is received at the device a visual and or audio signal is activated to indicate to the user that a card has arrived. The user may then view the compressed video message by using the video player software

Brief Description of Drawings

Preferred embodiments of the present invention are hereinafter described, by way of example only, with reference to the accompanying drawings, wherein:

Figure 1 is a simplified block diagram of an object oriented multimedia system of the preferred embodiment;

Figure 2 is a schematic diagram illustrating the three major packet types interleaved into an object oriented data stream of the preferred embodiment;

Figure 3 is a block diagram illustrating the three phases of data processing in an object oriented multimedia player of the preferred embodiment;

Figure 4 is a schematic diagram showing the hierarchy of object types in an object oriented data file of the preferred embodiment;

Figure 5 is a diagram showing a typical packet sequence in a data file or stream of the preferred embodiment;

Figure 6 is a diagram illustrating the information flow between client and server components of an object oriented multimedia player of the preferred embodiment;

Figure 7 is a block diagram showing the major components of an object oriented multimedia player client of the preferred embodiment;

Figure 8 is a block diagram showing the functional components of an object oriented multimedia player client of the preferred embodiment;

Figure 9 is a block diagram of a preferred embodiment of the client rendering engine;

Figure 10 is a block diagram of a preferred embodiment of the client interaction engine;

Figure 11 is a block diagram of the local server component of an interactive multimedia player of the preferred embodiment;

Figure 12 is a block diagram of a remote streaming server of the preferred embodiment;

Figure 13 is a block diagram of an object-oriented video encoder of the preferred embodiment;

Figure 14 is a block diagram of an input colour processing component of a video encoder of the preferred embodiment;

Figure 15 is a block diagram of the components of a region update selection process used in a video encoder of the preferred embodiment;

Figure 16 is a diagram of the tree splitting method used in a video encoder of the preferred embodiment;

Figure 17 is a diagram of three fast motion compensation methods used in a video encoder of the preferred embodiment;

Figure 18 is a block diagram of an ultra-thin computing client Local Area wireless Network (LAN) system of the preferred embodiment;

Figure 19 is a block diagram of an ultra-thin computing client Wide Area wireless Network (WAN) system of the preferred embodiment;

Figure 20 is a block diagram of an ultra-thin computing client Remote LAN server system of the preferred embodiment;

Figure 21 is a block diagram of an multiparty wireless videoconferencing system of the preferred embodiment;

Figure 22 is a block diagram of an interactive 'video on demand' system, with targeted user video advertising, of the preferred embodiment;

Figure 23 is a block diagram of an interactive 'video on demand' system of the preferred embodiment, with user authentication, access control, billing and usage metering;

Figure 24 is a block diagram of a video security/surveillance systems of the preferred embodiment; and

Figure 25 is a block diagram of an electronic greeting card system and service of the preferred embodiment.

Detailed Description of the Preferred Embodiment

General System Architecture

Typical video players present a passive experience to users. They read a single compressed video data stream and play it by performing a single, fixed decoding transformation on the data. In contrast, an object oriented video player, as described herein, provides advanced interactive video capabilities and allows dynamic composition of multiple video objects from multiple sources. This permits not only multiple video objects to coexist, but also determines what objects may coexist at any moment in real-time, based on either user interaction or predefined settings. For example, a scene in a video may be scripted to have one of two different actors perform different things in a scene depending on some user preference or user interaction.

To provide such flexibility, an object oriented video system has been developed including client and server components, as shown in Figure 1. The server components include an encoder 50 which compresses raw multimedia object data 51 into a compressed object data file 52, and a dynamic media composition engine 76 which multiplexes the data for each object from a number of sources together with definition and control data, and sends the resulting data stream to the player client. The player client includes a decoding engine 62 which uncompresses the object data stream and renders the various objects before sending them to the appropriate output devices 61. The decoding engine 62 performs operations on three interleaved streams of data: compressed data packets 64, definition packets 66, and control packets 68, as shown in Figure 2. The compressed data packets 64 contain the compressed object (*e.g.*, video) data to be decoded by an applicable encoder/decoder ('codec'). The methods for encoding and decoding video data are discussed in a later section. The definition packets 66 convey media format and other information that is used to interpret the compressed data packets 64. The control packets 68 define object behaviour, rendering, animation and interaction parameters.

As shown in Figure 3, three separate transforms are applied to the object oriented data to generate a final audio-visual presentation via a display 70 and an audio subsystem. A 'dynamic media composition' (DMC) process 76 modifies the actual content of the input stream and sends the data to the decoding engine 62. In the decoding engine 62, a normal decoding process 72 extracts the compressed audio and video data and sends it to a rendering engine 74 where other transformations are applied, including geometric transformations of rendering parameters for individual objects, (*e.g.*, translation). Each transformation is individually controlled through parameters inserted into the data stream.

The specific nature of each of the final two transformations depends on the output of the dynamic media composition process 76, as this determines the content of the data stream passed to the decoding engine 62. For example, the dynamic media composition process 76 may insert a specific video object into the bit stream. In this case, in addition to the video data to be decoded, the data bit stream will contain configuration parameters for the decoding process 72 and the rendering engine 74.

The data format of the object oriented bit stream permits seamless integration between different kinds of media objects, supports user interaction with these objects and enables programmable control of the content in a displayed scene, both when streaming the data from a remote server or accessing locally stored content. As shown in Figure 4, the data format defines a hierarchy of entities as follows: an object oriented data file 80 may contain one or more scenes, and each scene may contain one or more streams which contain one or more separate simultaneous media objects. These objects may be of a single media type such as video, audio, text or vector graphics, or composites of these. Multiple instances of each media type may simultaneously occur together with other media types in a single scene. Each object can contain one or more frames. When more than one media object is present in a scene, the frames for each are interleaved. A single media object is defined by a sequence of packets consisting of one or more definition packets, followed by data packets and any control packets all bearing the same object_id number. Further details of the file format are given in a later section.

In the case of download and play of video data, to allow interactive, object oriented manipulation of multimedia data, such as the ability to choose which actors appear in a scene, the input data must not consist of a single scene with a single "actor" object, but one or more alternative object data streams within each scene that may be selected or "composited-in" to the scene displayed at run-time based on user input. Since the composition of the scene is not known prior to runtime, it is not possible to interleave the correct object data streams into the scene. As shown in Figure 5, a stored scene consists of a number of separate selectable streams, one for each "actor" object that is a candidate for the dynamic media composition process. Only the first stream in a scene contains more than one (interleaved) media object. The first stream within a scene defines the scene structure, constituent objects and behaviour, and additional streams in a scene contain optional object data streams. A directory of streams is provided at the beginning of each scene to enable random access to each separate stream.

While the bit stream is capable of supporting advanced interactive video capabilities and dynamic media composition, it supports three implementation levels, providing various levels of functionality. These are:

1. Passive media: Single-object, non-interactive player

2. Interactive media: Single-object, limited interaction player
3. Object-oriented active media: Multi-object, fully interactive player

The simplest implementation provides a passive viewing experience with a single instance of media and no interactivity. This is the classic media player where the user is limited to playing, pausing and stopping the playback of normal video or audio.

The next level adds interaction support to passive media by permitting the definition of hot regions for click-through behaviour. This is provided by creating vector graphic objects with limited object control functionality. Hence the system is not literally a single object system, although it would appear so to the user. Apart from the main media object being viewed transparent, clickable vector graphic objects are the only other types of objects permitted. This allows simple interactive experiences to be created such as non-linear navigation, etc.

The final implementation level defines the unrestricted use of multiple objects and full object control functionality, including animations, conditional events, etc. and requires the implementation of all of the components in this architecture. In practice, the differences between this level and the previous may only be cosmetic.

The bit stream supports client side and server side interaction. Client side interaction is supported via a set of defined actions that may be invoked through objects which causes modification of the user experience. Server side interaction support is where user interaction is relayed to a remote server via a back channel, and provides mediation of the service/content provision to online users, predominantly in the form of dynamic media composition. Hence an interactive media player to handle the bit stream consists of a client-server architecture. The client is responsible for decoding data sent to it from the server, synchronisation, applying the rendering transformations, compositing the final display output, managing user input and forwarding user control back to the server. The server is responsible for managing, reading, and parsing partial bit streams from the correct source(s), constructing a composite bit stream based on user input with appropriate control instructions from the client, and forwarding the bit stream to the client for decoding and rendering. This server side DMC permits the content of the media to be composited in real-time, based on user interaction or predefined settings in a stored program script.

The media player supports both server side and client side interaction/functionality when playing back data stored locally and even when the data is being streamed from a remote server. Since it is the responsibility of the server to perform the DMC and manage sources, in the local playback case the server is co-located with the client, while being remotely located in the streaming case. Hybrid operation is also supported, where the client accesses data from both a local and a remotely located source/server.

Interactive Client

A player client, as shown in Figure 7, is able to receive and decode the data transmitted by the DMC process 16 of Figure 3. The player client includes a number of components to execute the decoding process. The steps of the decoding process are simplistic when compared to the encoding process, and can be executed entirely by software compiled on a low power mobile computing device. An input data buffer 30 is used to hold the incoming transmitted data until a full packet has been received or read. The data is then forwarded to an input data switch 32, either directly or via a decryption unit 34, to determine the sub-processes 38, 40, 42 required to decode the data and then forward the data to the correct component according to the packet type that executes that sub-process. Separate components 38 and 42 perform video, graphics, text and audio decoding. Both the video and audio decoding modules in the decoder independently decompress any data sent to them and perform a preliminary rendering into a temporary buffer. An object management component 40 extracts object behaviour and rendering information for use in controlling the video scene. A rendering engine 44 renders visual objects on the basis of data received from the video decoding and object management components 38, 40. An audio play back component 46 generates audio on the basis of data received from the audio decoding and object management component 40. A user input/control component 48 generates instructions and controls both the video and audio generated by the display and playback components 44 and 46. The user control component 48 also transmits control messages back to the video server.

A block diagram of the player client architecture is shown in Figure 8, including the following components:

1. ~~Codecs with optional data stores for the main data paths (components 38, 42)~~
2. Rendering engine (components 44 and 46 combined))
3. Interaction management engine (components 40 and 48 combined)
4. Object control path (part of component 40)
5. Input buffer and demultiplexing switch (components 30 and 32 combined).

There are two flows of data through the client system. The compressed bit stream comes from the server and is delivered to the demultiplexing switch, where the bit stream is split up into compressed data, definition and control packets. Each packet is individually routed to the appropriate codec, based on the packet type as identified in the packet header. Object control packets are sent up the object control path to be decoded. One codec instance and object store exists for each media object and for each media type. Hence there are not only different codecs for each media type, but if there are three video objects in a scene, then there will be three instances of video codecs. Each codec accepts the appropriate data packets sent to it and buffers the decoded data in the object data stores. Each object store is individually responsible for managing the synchronisation of each media object; if the decoding is lagging the (video) frame refresh rate, then the codec is instructed to drop frames as appropriate. The data in the object stores are read from by the rendering engine to composite the final displayed scene. Read and write access to the object data stores is asynchronous such that the codec may only update the data store at a slow rate, while the rendering engine may be reading that data at a faster rate, or vice versa, depending on the overall media synchronisation requirements. The rendering engine reads the data from each of the object stores and composes both the final display scene and the acoustic scene, based on rendering information from the interaction engine. The result of this process is a series of bitmaps that are handed over to the system graphical user interface to be displayed on the display device and a series of audio samples to be passed to the system audio device.

The secondary data flow through the client system comes from the user via the GUI to the interaction engine, where it is split up, with some of it being passed to the rendering engine in the form of rendering parameters, and the rest being passed back through the back channel to the server to control the dynamic media composition engine. The operation of the interaction engine is controlled by the object control data sent from the server that defines how it must interpret user events from the GUI, and what animations and interactive behaviours are associated with individual media objects. The interaction engine is responsible for controlling the rendering engine to carry out all necessary rendering transformations.

The rendering engine has three main components as shown in Figure 9. A bitmap compositor reads bitmaps from the visual object store buffers and composites them into the final scene raster. A vector graphic scan converter renders the display list from the vector graphic codec onto the display scene raster. An audio mixer reads the audio store buffers and mixes the audio data together, before passing the result to the audio device. The sequence in which the various object store buffers are read and how their content is transformed onto the display scene raster is determined by rendering parameters from the interaction management engine. Possible transformations include Z-order, 3D orientation, position, scale, transparency, colour, and volume. To speed up the rendering process, it may not be necessary to render the entire display scene, but only a portion of it.

The display scene must be rendered whenever visual data is received from the server according to synchronization information, when a user selects a button by clicking or drags an object that is draggable, and when animations are updated. To render the scene, it must be composited into an offscreen buffer (the display scene raster), and then drawn to the output device. The rendering engine keeps a linked list that contains a pointer to each media object store containing visual objects that is sorted according to Z order. To render the display scene, the bitmap compositor reads each video object store in sequence according to the Z-order associated with each media object in back to front order and copies it to the display scene raster. Z order increases towards the viewer such that objects with higher Z order values are drawn over ones lower depth values. Hence, the lower Z order objects must be drawn before higher Z order objects. If no Z order has been assigned to objects, the z order value for an object can be obtained by multiplying the object_ID value by 255. If two have the same Z order, they are drawn in order of ascending object IDs.

In the process of performing the bitmap copy operation from the object store to the display scene raster, the appropriate 2/3D geometric transform is applied to determine which coordinate pixels must be mapped to. After the geometric transform, an alpha blending composition process is performed to set the defined level of transparency for the object. However, unlike traditional alpha blending processes that need to separately encode the mixing factor for every pixel in a bitmap, this approach does not make use of an alpha channel. Instead, it utilizes a single alpha value specifying the degree of opacity of the entire bitmap in conjunction with embedded indication of transparent regions in the actual bitmap representation. The composition makes use of the three region types that the video frame can have: colour pixels to be rendered, areas to be made transparent, and areas to remain unchanged. The colour pixels are appropriately alpha blended into the scene display, and the unchanged pixels are ignored so the display scene is unaffected. The transparent pixels force the corresponding background display scene pixel to be refreshed. This can be performed when the pixel of the object in question is overlaying some other object by simply doing nothing, but if the pixel is being drawn directly over the scene background, then that pixel needs to be set to the scene background colour.

If the object store contains a display list in place of a bitmap, then the geometric transform is applied to each of the coordinates in the display list, and the alpha blending is performed during the scan conversion of the graphics primitives specified within the display list.

The bitmap compositor must be able to support display scene rasters with different colour resolutions, and manage bitmaps with different bit depths. If the display scene raster has a depth of 15,16 or 24 bits, and a bitmap is a colour mapped 8 bit image, then the bitmap compositor reads each colour index value from the bitmap, looks up the colour in the colour map associated with that particular object store, and writes the red, green and blue components of the colour in the correct format to the display scene raster. If the bitmap is a continuous tone image in this case, the bitmap compositor simply copies the colour value of each pixel into the correct location on the display scene raster. If the display scene raster has a depth of 8 bits and a colour look up table, the approach taken depends on the number of objects displayed. If only one video object is being displayed, then its colour map is copied directly into the display scene raster's colour map. If multiple video objects exist, then the display scene raster will be set up with a generic colour map, and the pixel value set in the display scene raster will be the closest match to the colour indicated by the index value in the bitmap.

The rendering engine is also responsible for evaluating when a user has selected a visual object on the screen by comparing the pen event location coordinates with each object displayed. This 'hit testing' is requested by the interaction control unit. Hit testing requires an inverse geometric transformation of the pen event location for each object, and then evaluation of the transparency of the bitmap at the resulting inverse transformed coordinate. If the evaluation is true, a hit is registered, and the result is returned to the interaction control unit.

The audio mixer reads each audio frame stored in the relevant object store in round robin fashion, and mixes the audio data together according to volume controls from the interaction engine to obtain the composite frame. It then passes the mixed audio data to the audio output device.

The object control path is basically a codec that reads the coded object control packets from the input stream and issues the indicated control instructions to the interaction management engine. Control instructions may be issued to change individual objects or system wide attributes. These controls are wide ranging, and include rendering parameters, definition of animation paths, creating conditional events, controlling the sequence of media play, assigning hyperlinks, setting and resetting system state registers, etc, and defining user-activated object behaviours.

The interaction engine has to manage a number of different processes, as shown in Figure 10. The interaction engine may immediately perform predefined actions unconditionally, or delay execution until some system conditions are met, or it may respond to user input with a defined behaviour either unconditionally or subject to system conditions. Possible actions include rendering attribute changes, animations, looping and non-sequential play sequences, jumping to hyperlinks, dynamic media composition where a displayed object stream is replaced by another object, and other system behaviours that are invoked when given conditions or user events become true. The interaction engine has no predefined behaviour: all of the actions and conditions that the interaction management engine may perform or respond to are defined by ObjectControl packets.

The interaction engine consists of three main components: the object control logic, the waiting actions manager, and the animation manager. The animation manager stores all animations that are currently in progress. For each active animation, the manager interpolates the rendering parameters at intervals specified by the object control logic that are sent to the rendering engine. When an animation has completed, it is removed from the list of active animations unless it is defined to be a looping animation. The waiting actions manager stores all object control actions to be applied subject to a condition becoming true. The object control regularly polls the waiting actions manager and evaluates the conditions associated with each waiting action. If the conditions for an action are met, the object control will execute an action and purge it from the list, unless the action has been defined as an object behaviour, in which case it remains on the list for further future executions.

In addition, ObjectContol packets may set a number of user-definable system flags. These are used to permit the system to have a memory of its current state. For example, one of these flags may be set when a certain scene or frame in the video is played, or when a user interacts with an object. This may then be used to determine what scene to play next in nonlinear videos, or what objects to render in a scene. In an e-commerce application, the user may drag one or more iconic video objects onto a shopping basket object. This will then register the intended purchases. When the shopping basket is clicked, the video will jump to the checkout scene, where a list of all of the objects that were dragged onto the shopping basket appears, permitting the user to confirm or delete the items. A separate video object can be used as a button, indicating that the user wishes to register the purchase order or cancel it.

ObjectControl packets may contain conditions that must be satisfied for any specified actions to be executed. Conditions may include the system state, local or streaming playback, system events, specific user interactions with objects, etc. A condition may have the wait flag set, indicating that if the condition isn't currently satisfied, then wait until it is. The wait flag is often used to wait for user events such as penUp. When a waiting action is satisfied, it is removed from the list of waiting actions associated with an object. If the behaviour flag of an ObjectControl packet is set, then the action will remain with an object in the waiting actions list even after it has executed.

An ObjectControl packet may specify that the action is to affect another object. In this case, the conditions must still be satisfied on the object specified in the base header, but the action is executed on the other object.

An ObjectControl packet may have the `animation` flag set, indicating that multiple commands will follow rather than a single command (such as moveto). If the `animation` flag isn't set, then the actions are executed as soon as the conditions are satisfied. As often as any rendering changes occur, the display scene must be updated. Unlike most rendering actions that are driven by either user events or control packets, animations must force rendering updates themselves. After the animation is updated, it must be checked as to whether the entire animation is complete and, if so, removes it from the list of animations. The animation path interpolator must determine where, between which two control points, the animation is currently positioned. This information, along with a ratio of how far the animation has progressed between the two control points (the 'tweening' value), is used to interpolate the rendering relevant parameters. The tween value is expressed as a ratio in terms of a numerator and denominator:

$$X = x[start] + (x[end] - x[start]) * numerator / denominator$$

If the animation is set to loop, then the start time of the animation is set to the current time when the animation has finished, so that it isn't removed after the update.

The client must allow for the following types of high-level user interaction: clicking, dragging, overlapping, and moving. An object may have a button image associated with it that is displayed when the pen is held down over an object. If the pen is moved a specified number of pixels when it is down over an object, then the object is dragged (as long as dragging isn't protected by the object or scene). Dragging actually moves the object under the pen. When the pen is released, the object is moved to the new position unless moving is protected by the object or scene. If moving is protected, then the dragged object moves back to its original position when the pen is released. Dragging may be enabled so that users can drop objects on top of other objects (*e.g.*, dragging an item onto a shopping basket). If the pen is released whilst the pen is also over other objects, then these objects are notified of an overlap event with the dragged object.

Objects may be protected from clicks, moving, dragging, or changes in transparency or depth through ObjectControl packets. A PROTECT command may have individual object scope or system scope. If it has system scope, then all objects are affected by the PROTECT command. System scope protection overrides object scope protection.

The JUMPTO command has four variants. One permits jumping to a new given scene in a separate file specified by a hyperlink, another permits replacing a currently playing media object stream in the current scene with another media object from a separate file or scene specified by a hyperlink, and the other two variants permit jumping to a new scene within the same file or replacing a playing media object with another within the same scene specified by directory indexes. Each variant may be called with or without an object mapping.

While most of the interaction control functions can be handled by the client using the rendering engine in conjunction with the interaction manager, some control instances must be handled at a lower level and are passed back to the server. This includes all commands for non-linear navigation, such as jumping to hyperlinks and dynamic scene composition.

Server Software

The purpose of the server system is to create the correct data stream for the client to decode and render. The content of this stream is a function of the dynamic media composition process and non-sequential access requirements imposed by non-linear media navigation. The source data for the composite data stream may come from either a single source or from multiple sources. In the single source case, the source must contain all of the optional data components that may be required to composite the final data stream. Hence this source is likely to contain a library of different scenes, and multiple data streams for the various media objects that are to be used for composition. Since these media objects may be composited simultaneously into a single scene, advanced non-sequential access capabilities are required on the part of the server to select the appropriate data components from each media object stream in order to interleave them into the final composite data stream to send to the client. In the multiple source case, each of the different media objects to be used in the composition can have individual sources. Having the component objects for a scene in separate sources relieves the server of the complex access requirements, since each source need only be sequentially accessed, although there are more sources to manage.

Both source cases are supported. For download and play functionality, it is preferable to deliver one file containing the packaged content, rather than multiple data files. For streaming play, it is preferable to keep the sources separate, since this permits much greater flexibility in the composition process and permits it to be tailored to specific user needs such as targeted user advertising. The separate source case also presents a reduced load on server equipment since all file accesses are sequential. Hence, standalone players need a local client system and a local single source server system, while streaming players need a local client system and a remote multi-source server. However, a player is also able to play local files and streaming content simultaneously, so the client system is also able to simultaneously accept data from both a local server and a remote server.

In the simplest case with passive media playback, the local server opens a data file and sequentially reads its contents, passing the data to the client. Upon a user command, this file reading operation may be stopped, paused, continued from its current position, or restarted from the beginning of the file. In this simple case there, the server performs two functions: accessing the data source, and controlling this access. These can be generalised into the data source manager and the dynamic media composition engine, as shown in Figure 11.

In the more advanced case with local playback of video and dynamic media composition, it is not possible for the client to merely sequentially read one predetermined stream with multiplexed objects, because the contents of the multiplexed stream are not known when the file is created. Therefore, the local data file includes multiple streams for each scene stored contiguously within the file. The server randomly accesses each stream within a scene and selects the objects which need to be sent to the client for rendering.

The data source manager/multiplexer randomly accesses the source file and reads the correct data and control packets from the various streams in the file required to compose the display scene, and multiplexes these together to create the composite data stream that the client uses to render the composite scene. A stream is purely conceptual as there is no packet indicating the start of a stream. There is, however, an end of stream packet to demarcate stream boundaries. Typically, the first stream in a scene contains descriptions of the objects within the scene. Object control packets within the scene may change the source data for a particular object to a different stream. The server then needs to read more than one stream simultaneously from within a file when performing local playback. Rather than creating separate threads, an array or linked list of streams can be created. The data stream manager reads one packet from each stream in a round robin fashion. At a minimum, each stream needs to store the current position in the file, and also a list of referencing objects.

In this case, the dynamic media composition engine, upon the receipt of user control information from the client, selects the correct combination of objects to be composited together, and ensures that the data stream manager knows where to find these objects, based on directory information provided by the data stream manager. This may also require an object mapping function to map the storage object identifier with the run time object identifier that can differ depending upon the composition. A typical situation where this may occur is when multiple scenes in a file may wish to share a particular video or audio object. Since a file may contain multiple scenes, this can be achieved by storing shared content in a special "library" scene. Objects within a scene have object IDs ranging from 0-200, and every time a new SceneDefinition packet is encountered, the scene is reset with no objects. Each packet contains a base header that specifies the type of the packet as well as the object ID of the referenced object. An object ID of 254 represents the scene, whilst an object ID of 255 represents the file. When multiple scenes share an object data stream, it is not known what object IDs will have already been allocated for different scenes; hence, it is not possible to preselect the object IDs in the shared object stream as these may already be allocated in a scene. One way to get around this problem is to have unique IDs within a file, but this increases storage space and makes it more difficult to manage sparse object IDs. The problem is solved by allowing each scene to use their own object IDs and when a packet from one scene indicates a jump to another scene, it specifies an object mapping between IDs from each scene. When packets are read from the new scene, the mapping is used to convert the object IDs.

Object mapping information is expected to be in the same packet as a JUMPTO command. If this information is not available, then the command is simply ignored. Object mappings are represented using two arrays. One for the source object IDs which will be encountered in the stream, and the destination object IDs which the source object IDs will be converted to. If an object mapping is present in the current stream, then the destination object IDs of the new mapping must be converted using the object mapping arrays of the current stream. If an object mapping is not specified in the packet, then the new stream inherits the object mapping of the current stream (which may be `null`). All object IDs within a stream must be converted. For example, parameters such as: base header IDs, other IDs, button IDs, copyFrame IDs, and overlapping IDs must all be converted into the destination object IDs.

In the remote server scenario, the server is remote from the client, so that data must be streamed to the client. The media player client is designed to decode packets received from the server and to send back user operations to the server. In this case, it is the remote server's responsibility to respond to user operations (such as clicking an object), and to modify the data being sent to the client. In this case, each scene contains only a single multiplexed stream (composed of one or more objects).

In this scenario, the server composes scenes in real-time by multiplexing multiple object data streams, based on client requests to construct a single multiplexed stream (for any given scene) that is streamed to the client for playback. This architecture allows the media content being played back to change, based on user interaction. For example, two videos may be playing simultaneously. When the user clicks or taps on one, it changes to a different video clip, whilst the other video remains unchanged. Each video may come from a different source, so the server opens both sources and interleaves the bit streams, adding appropriate control information and forwarding the new composite stream to the client. It is the server's responsibility to modify the stream appropriately before streaming it to the client.

As shown in Figure 12, the remote server has the same two main functional components as the local server: the data source manager and the dynamic media composition engine. However, the remote server multiplexer can take input from multiple source manager instances with single inputs and from the dynamic media composition engine, instead of from a single source manager with multiple inputs. Along with the object data packets that are multiplexed together from the source(s), the multiplexer inserts additional control packets into the data stream to control the rendering of the component objects in the composite scene. The remote stream managers are also simpler, as they only perform sequential access. In addition to this, the remote server includes an XML parser to enable programmable control of the dynamic media composition through IAVML scripting. The remote server also accepts a number of inputs from the server operator to further control and customize the dynamic media composition process. Possible inputs include the time of day, day of the week, day of the year, geographic location of the client, a user's demographic data such as gender, age, any stored user profiles, etc. These inputs can be utilized in an IAVML script as variables in conditional expressions. The remote server is also responsible for passing user interaction information such as object selections and form data back to the server operator's database for later follow up processing such as data mining, etc.

Video Decoder

It is inefficient to store, transmit and manipulate raw video data, and so computer video systems normally encode video data into a compressed format. The section following this one describes how video data is encoded into an efficient, compressed form. This section describes the video decoder, which is responsible for generating video data from the compressed data stream. The video codec supports arbitrary shaped video objects. It represents each video frame using three information components: a colour map, a tree based encoded bitmap, and a list of motion vectors. The colour map is a table of all of the colours used in the frame, specified in 24 bit precision with 8 bits allocated for each of the red, green and blue components. These colours are referenced by their index into the colour map. The bitmap is used to define three things: the colour of pixels in the frame to be rendered on the display, the areas of the frame that are to be made transparent, and the areas of the frame that are to be unchanged. All of the pixels in each encoded frame must be allocated to one of these three functions. Which of these roles a pixel has is defined by its value, since the last two values in the possible range of colour values are reserved for the last two functions. For example, if an 8 bit colour representation is used, then colour value 0xFF may be assigned to indicate that the corresponding on screen pixel is not to be changed from its current value, and the colour value of 0xFE may be assigned to indicate that the corresponding on screen pixel for that object is to be transparent. The final colour of an on screen pixel, where the encoded frame pixel colour value indicates it is transparent, depends on the background scene colour and any underlying video objects. The specific encoding used for each of these components that makes up an encoded video frame is described below.

The colour table is encoded by first sending an integer value to the bit stream to indicate the number of table entries to follow. Each table entry to be sent is then encoded by first sending its index, since only selected table entries may be sent at any one time. Following this, a one bit flag is sent for each colour component (Rf, Gf and Bf) indicating if it is ON that the colour component is being sent as a full byte and if the flag is OFF that only the high order nibble (4 bits) of the respective colour component will be sent and the low order nibble is set to zero. Hence the table entry is encoded in the following pattern where the number or C language expression in the parenthesis indicates the number of bits being sent: R(Rf?8:4?), G(Rf? 8: 4), B(Rf?8: 4).

The motion vectors are encoded as an array. First, the number of motion vectors in the array is sent as a 16 bit value, followed by the size of the macro blocks, and then the array of motion vectors. Each the entry in the array contains the location of the macro block and the motion vector for the block. The motion vector is encoded as two signed nibbles, one each for the horizontal and vertical components of the vector.

The actual video frame data is encoded using a preordered tree traversal method. There are only two types of leaves in the tree: transparent leaves, and region colour leaves. The transparent leaves indicate that the onscreen displayed region indicated by the leaf will not be altered, while the colour leaves will force the onscreen region to the colour specified by the leaf. In terms of the three functions that can be assigned to any encoded pixel as previously described, the transparent leaves would correspond to the colour value of 0xFF while pixels with a value of 0xFE indicating that the on screen region is to be forced to be transparent are treated as normal region colour leaves. The encoder starts at the top of the tree and for each node stores a single bit to indicate if the node is a leaf or a parent. If it is a leaf the value of this bit is set to ON and another single bit is sent to indicate if the region is transparent (OFF) otherwise it is set to ON followed by a another one bit flag to indicate if the colour of the leaf is sent as an index into a FIFO buffer or as the actual index into the colour map. If this flag is set to OFF then a two bit codeword is sent as the index of one of the FIFO buffer entries. If the flag is ON, this indicates that the leaf colour is not found in the FIFO, and the actual colour value is sent and also inserted into the FIFO, pushing out one of the existing entries. If the tree node was a parent node, then a single OFF bit is stored, and each of the four child nodes are then individually stored using the same method. When the encoder reaches the lowest level in the tree, then all nodes must be leaf nodes and the leaf/parent indication bit is not used, instead storing first the transparency bit followed by the colour codeword. The pattern of bits sent can be represented as shown below. The following symbols are used: node type (N), transparent (T), FIFO Predicted colour (P), colour value (C), FIFO index (F)

```
N(1) ---off → N(1)[...], N(1)[...], N(1)[...] , N(1)[...]
     \---- on → T(1) --- off
                   \------on → P(1)  --- off → F(2)
                                     \--- on → C(x)
```

The decoding process involves first reading the colour map values from the data stream, and updating its local colour map with the values. Next, the motion vectors are read, decoded and used to copy the indicated macro blocks from the previously buffered frame to the new locations indicated by the vectors. Following this, the encoded bitmap tree data is read, decoded and the region values in the frame are modified according to the leaf values. They may be overwritten with new colours, set to transparent or left unchanged.

Video Encoder

To this point, the discussion has focussed on the manipulation of pre-existing video objects and files which contain video data. The previous section described how compressed video data is decoded to produce raw video data. In this section, the process of generating this data is discussed.

A video encoder, shown in Figure 13, executes an object-oriented video encoding process. The system is designed to support a number of different codecs. One such codec is described here. The encoder process comprises eight main components. The components can be implemented in software, but to enhance the speed of the encoder, all the components are implemented in an application specific integrated circuit (ASIC) developed specifically to execute the steps of the encoding process. An input colour processing component 10 receives and processes individual input video frames and eliminates redundant and unwanted colours. This also removes noise from video images. An audio coding component 12 compresses input audio data using adaptive delta pulse code modulation (ADPCM) according to ITU specification G.723. A scene/object control data component 14 encodes scene animation and presentation parameters associated with the input audio and video and which determine the relationships and behaviour of each input video object. A colour difference management and synchronisation component 16 receives the output of the input colour processor 10, and determines the encoding required using the previously encoded frame as a basis. The output is then provided to a combined spatial/temporal coder 18 to compress the video data. The output is provided to a decoder 20, which executes the inverse function to provide the frame to the colour management and synchronisation component 16 after a one frame delay 24. A transmission buffer 22 receives the output of decoder 18, the audio coder 12 and the control data component 14. The transmission buffer 22 manages transmission from a video server housing the encoder, by interleaving and controlling data rates. If necessary, the encoded data can be encrypted by an encryption component 28 for transmission.

The input colour processing component 10 performs reduction of statistically insignificant colours. The colour space chosen to perform this colour reduction is unimportant as the same outcome can be achieved using any one of a number of different colour spaces. The colour management component 16 manages the colour data generated by the first two processes of the component 10.

The reduction of the statistically insignificant colours may be implemented using various vector quantisation techniques including popularity, median cut, k-nearest neighbour and variance methods. The choice of method is made to maintain the highest amount of time correlation between the quantised video frames. The input to this process is the candidate video frame, and the process proceeds by analysing the statistical distribution of colours in the frame. The output of the vector quantisation process is a table of representative colours for the entire frame that can be limited in size. In the case of the popularity methods, the most frequent N colours are selected. Finally, each of the colours in the original frame is remapped to one of the colours in the representative set..

Following the input colour processing 10, the next component of the video encoder takes the now indexed colour frames and performs motion compensation. The preferred method starts by segmenting the video frame into small blocks and determining all blocks in a video frame where the number of pixels needing to be replenished or updated and are not transparent exceed some threshold. The motion compensation process is then performed on the resultant pixel blocks. First, a search is made in the neighbourhood of the region to determine if the region has been displaced from the previous frame. One way to accomplish this is to calculate the mean square error or sum square error metric between the reference region and a candidate displacement region. This process can be performed using an exhaustive search or one of a number of other existing search techniques, such as the 2D logarithmic, three step and simplified conjugate direction search, as shown in Figure 15. The aim of this search is to find the displacement vector for the region often called the motion vector. This is found by locating the displacement for a region where the number of pixels that are different in the previous frame compared to the current frame region are the least if the region is not transparent. Once the motion vector is found, then the region is motion-compensated by predicting the value of the pixels in the region from their original location in the previous frame according to the motion vector. The motion vector may be zero if the vector giving the least difference corresponds to no displacement. The motion vector for each displaced block is encoded into the output bitstream. Following this, the perceptual difference is calculated between the motion compensated previous frame and the current frame.

The colour management component 16 of the input colour processing manages all colour changes in the video and is responsible for Region Update Selection. The colour data produced by the prior stages in the colour processing results in a table containing a set of all displayed colours. This set of colours changes dynamically, given that it is created by an adaptive process on a per frame basis. This permits the colour composition of the video frames to change without reducing the image quality. Selecting an appropriate scheme to manage the adaptation of the colour map is important. Three distinct possibilities exist for the colour map: it may be static, segmented and partially static, or fully dynamic. With a fixed or static colour map, the local image quality will be reduced, but high correlation is preserved from frame to frame, leading to high compression gains. In order to maintain high quality images for video where scene changes may be frequent, the colour map must be able to adapt instantaneously. Selecting a new optimal colour map for each frame has a high bandwidth requirement, since not only must the entire colour map be replaced every frame, but also a large number of pixels in the image would need to be remapped each time. This remapping also introduces the problem of colour map flashing. A compromise is to only permit limited colour variations between successive frames. This can be achieved by partitioning a colour map into static and dynamic sections, or by limiting the number of colours that are allowed to vary per frame. In the first case, only the entries in the dynamic section of the table can be modified, which ensures that certain predefined colours will always be available. In the other scheme, there are no reserved colours and any may be modified. While this approach helps preserve some data correlation, the colour map may not be able to adapt quickly enough in some cases to eliminate image quality degradation. Existing approaches compromise image quality to preserve frame-to-frame image correlation.

For any of these dynamic colour map schemes, synchronisation is required to preserve temporal correlations. This synchronisation process has three components:

1. Ensuring that colours carried over from each frame into the next are mapped to the same indexes over time. This involves resorting each new colour map in relation to the current one.

2. A replacement scheme is used for updating the changed colour map. To reduce the amount of colour flashing, the most appropriate scheme is to replace the obsolete colour with the most similar new replacement colour.

3. Finally, all existing references in the image to any colour which is no longer supported are replaced by references to currently supported colours.

The colour difference management component 16 is also responsible for calculating the perceived colour difference at each pixel between the current and preceding frame. This perceived colour difference is based on a similar calculation to that described for the perceptual colour reduction. Pixels must be updated if their colour has changed more than a given amount. The colour difference management component 16 is also responsible for purging all invalid colour map references in the image, and replacing these with valid references. Invalid colour map references may occur as old colours in the colour map are displaced by newer colours. This information is then passed onto the spatial/temporal coding component 18 in the video encoding process. This information indicates which regions in the frame are fully transparent, and which need to be replenished, and which colours in the colour map need to be updated. All transparent regions in a frame are identified by setting the value of the pixel to a predetermined value that has been selected to represent transparency. The inclusion of this value permits the creation of arbitrarily shaped video objects. To ensure that prediction errors do not accumulate and degrade the image quality, a loop filter is used. This forces the frame replenishment data to be determined from the present frame and the accumulated previous transmitted data (the current state of the decoded image), rather than from the present and previous frames. Figure 15 provides a more detailed view of the colour difference management component 16. The current frame store contains the resultant image from the input colour processing component 10. The previous frame store contains the frame buffered by the 1 frame delay component 24. The colour difference management component 16 is portioned into its two main components: the calculation of perceived colour differences between pixels, and cleaning up invalid colour map references. The perceived colour differences are evaluated with respect to a threshold to determine which pixels need to be updated and the resultant pixels are optionally filtered to reduce the data rate. The final update image that is sent to the spatial encoder is made up from the output of the spatial filter and the invalid colour map references.

This results in new replenishment data that must now be encoded. The spatial encoder uses a tree splitting method to recursively partition each frame into smaller polygons according to a splitting criteria. A quad tree split method used, as is shown in Figure 16. In one instance, that of zeroth order interpolation, this attempts to represent the image by a uniform block, the value of which is equal to the global mean value of the image. In another instance, first or second order interpolation may be used. If, at some locations of the image, the difference between this representative value and the real value exceeds some tolerance threshold, then the block is recursively subdivided uniformly, into two or four subregions, and a new mean is calculated for each subregion. For lossless image encoding, there is no tolerance threshold. Potentially every pixel in the image must be explicitly encoded and data expansion may occur instead of compression. The tree structures are composed of nodes and pointers, where each node represents a region and contains pointers to any child nodes representing subregions which may exist. There are two types of nodes: leaf and non-leaf nodes. Leaf nodes are those that are not further decomposed and as such have no children, instead containing a representative value for the implied region. Non-leaf nodes do not contain a representative value, since these consist of further subregions and as such contain pointers to the respective child nodes. These can also be referred to as parent nodes.

Dynamic Bitmap (Colour) Encoding

The actual video frame data is encoded using a preordered tree traversal method. There are actually two types of leaves in the tree: transparent leaves, and region colour leaves. The transparent leaves indicate that the region indicated by the leaf is unchanged from its previous value, and the colour leaves contain the region colour. In the case of zeroth order interpolation, the encoder starts at the top of the tree, and for each node stores a single ONE bit if the node is a leaf, or a ZERO if it is a parent. If the node was a leaf, then if the region is transparent, another single ZERO bit is stored, otherwise another single bit is stored to indicate if the colour is explicitly encoded or preferably if it is specified as a two bit codeword representing the colour of the leaf as an index into a FIFO buffer. This codeword indexes one of four entries in the FIFO buffer. If the colour is not available in this FIFO, then the actual colour value is stored into the bitstream and also inserted into the FIFO, pushing out one of the existing entries. The FIFO provides a window into the history of recent leaf colours. For example, index values of 00, 01 and 10 specify that the leaf colour is the same as the previous leaf, the previous different leaf colour before that, and the previous one before that respectively. If the node was a parent node and a ZERO bit was stored, then each of the four child nodes are then recursively stored as described. When the encoder reaches the lowest level in the tree, then all nodes must be leaf nodes and the leaf/parent indication bit is not used, instead storing first the transparency bit followed by the colour codeword.

The colourmap has similarly been compressed. The standard representation is to send each index followed by 24 bits, 8 to specify the red component value, 8 for the green component and 8 for the blue. In the compressed format, a single bit flag to indicate if each colour component is specified as a full 8 bit value or just as the top nibble with the bottom 4 bits set to zero. Following this flag, the component value is sent as 8 or 4 bits depending on the flag.

Encoding of Colour Prequantisation Data

For improved image quality, a first or second order interpolation can be used. This is performed by storing in the tree not only the mean colour for the region represented by each leaf, but also colour gradient information at each leaf. Reconstruction is then performed using quadratic or cubic interpolation to regenerate a continuous tone image. This may create a problem when displaying continuous colour images on devices with indexed colour displays. In these situations, the need to quantise the output down to 8 bits and index it in real time is prohibitive. In this case pre-calculated colour quantisation information can be encoded and sent with the encoded image to enable the decoder/player to apply the pre-calculated colour quantisation in real-time. This technique can also be used when reconstruction filtering is used that generates a 24 bit result that must be displayed on 8 bit devices. This problem can be resolved by sending a small amount of information to the decoder that describes the mapping from the 24 bit colour result to the 8 bit colour table. This information consists effectively of a three-dimensional array of size 32x64x32, where the cells in the array contain the index values for each r,g,b coordinate. Clearly storing and sending a total of 32x64x32 = 65,536 index values is a large overhead that makes the technique impractical. The solution is to encode this information in a compact representation. One method to do this is to encode this three dimensional array of indexes using an octree representation. This method takes the cube and recursively splits it in a similar manner to the quadtree based representation. Since the colour map is free to change dynamically, this mapping information must also be updated to reflect the changes in the colour map from frame to frame. A similar conditional replenishment method is proposed to perform this using the index value 255 to represent an unchanged coordinate mapping and other values to represent update values for the 3D mapping array. Like the spatial encoder, the process uses a preordered octree tree traversal method to encode the colour space mapping into the colour table. Transparent leaves indicate that the region of the colour space indicated by the leaf is unchanged and index leaves contain the colour table index for the colour specified by the coordinates of the cell. The octree encoder starts at the top of the tree and for each node stores a single ONE bit if the node is a leaf, or a ZERO bit if it is a parent. If it is a leaf and the colour space area is unchanged then another single ZERO bit is stored otherwise the corresponding colour map index is explicitly encoded as a n bit codeword. If the node was a parent node and a ZERO bit was stored, then each of the eight child nodes are recursively stored as described. When the encoder reaches the lowest level in the tree, then all nodes must be leaf nodes and the leaf/parent indication bit is not used, instead storing first the unchanged bit followed by the colour index codeword. The decoder performs the reverse process to reconstruct the 3D array from the encoded octree as in the 2D quadtree decoding process described. Then for any 24 bit colour value the corresponding colour index can be determined by simply looking up the index value stored in the 2D array. This technique can be used for mapping any non-stationary three-dimensional data onto a single dimension. This is normally a requirement when vector quantisation is used to select a code book that will be used to represent an original multidimensional data set. It does not matter at what stage of the process the vector quantisation is performed. For example, we could directly quadtree encode 24 bit data followed by VQ or we could VQ the data first and then quadtree encode the result as we do here.

The object scene control data input to the component 14 permits each object to be associated with one visual data stream, one audio data stream and one of any other data streams. It also permits various rendering and presentation parameters for each object to be dynamically modified from time to time throughout the scene. These include the amount of object transparency, object scale, object volume, object position in 3D space, and object orientation (rotation) in 3D space.

The compressed video and audio data is now transmitted or stored for later transmission as a series of data packets. There are a plurality of different packet types. Each packet consists of a common base header and the payload. The base header identifies the packet type, the total size of the packet including payload, what object it relates to and a sequence identifier. The following types of packets are currently defined: SCENEDEFN, VIDEODEFN, AUDIODEFN, TEXTDEFN, GRAFDEFN, VIDEODAT, AUDIODAT, TEXTDAT, GRAFDAT, OBJCTRL, SYSCTRL, USERCTRL, METADATA, DIRECTORY, STREAMEND. As described earlier, there are three main types of packets: definition, control and data packets. The control packets (CTRL) are used to define object rendering transformations, animations and actions to be executed by the object control engine, interactive object behaviours, dynamic media composition parameters and conditions for execution or application of any of the preceding for either individual objects or for entire scenes being viewed. The data packets contain the compressed information that makes up each media object. The format definition packets (DEFN) convey the configuration parameters to each codec, and specify both the format of the media objects and how the relevant data packets are to be interpreted. The scene definition packet defines the scene format, specifies the number of objects, and defines other scene properties. The USERCTRL packets are used to convey user interaction and data back to a remote server using a backchannel, the METADATA packets contain metadata about the video, the DIRECTORY packets contain information to assist random access into the bit stream, and the STREAMEND packets demarcate stream boundaries.

Access Control and Identification

Another component of the object oriented video system is means for encrypting/decrypting the video stream for security of content. The key to perform the decryption is separately and securely delivered to the end user, perhaps by encoding it using the RSA public key system.

An additional security measure is including a universally unique brand/identifier in an encoded video stream. This could take at least four principal forms:

a. In a videoconferencing application, a single unique identifier is applied to all instances of the encoded video streams

b. In broadcast video-on-demand (VOD) with multiple video objects in each video data stream, each separate video object has a unique identifier for the particular video stream

c. A wireless, ultrathin client system has a unique identifier which identifies the encoder type as used for wireless ultrathin system server encoding, as well as identifying a unique instance of this software encoder.

up the index value stored in the 2D array. This technique can be used for mapping any non-stationary three-dimensional data onto a single dimension. This is normally a requirement when vector quantisation is used to select a code book that will be used to represent an original multidimensional data set. It does not matter at what stage of the process the vector quantisation is performed. For example, we could directly quadtree encode 24 bit data followed by VQ or we could VQ the data first and then quadtree encode the result as we do here.

The object scene control data input to the component 14 permits each object to be associated with one visual data stream, one audio data stream and one of any other data streams. It also permits various rendering and presentation parameters for each object to be dynamically modified from time to time throughout the scene. These include the amount of object transparency, object scale, object volume, object position in 3D space, and object orientation (rotation) in 3D space.

The compressed video and audio data is now transmitted or stored for later transmission as a series of data packets. There are a plurality of different packet types. Each packet consists of a common base header and the payload. The base header identifies the packet type, the total size of the packet including payload, what object it relates to and a sequence identifier. The following types of packets are currently defined: SCENEDEFN, VIDEODEFN, AUDIODEFN, TEXTDEFN, GRAFDEFN, VIDEODAT, AUDIODAT, TEXTDAT, GRAFDAT, OBJCTRL, SYSCTRL, USERCTRL, METADATA, DIRECTORY, STREAMEND. As described earlier, there are three main types of packets: definition, control and data packets. The control packets (CTRL) are used to define object rendering transformations, animations and actions to be executed by the object control engine, interactive object behaviours, dynamic media composition parameters and conditions for execution or application of any of the preceding for either individual objects

d. A wireless ultrathin client system has a unique identifier that uniquely identifies the client decoder instance in order to match the Internet-based user profile to determine the associated client user.

The ability to uniquely identify a video object and data stream is particularly advantageous. In videoconference applications, there is no real need to monitor or log the teleconference video data streams, except where advertising content occurs (which is uniquely identified as per the VOD). The client side decoder software logs viewed decoded video streams (identifier, duration). Either in real time or at subsequent synchronisation, this data is transferred to an Internet-based server. This information may be used in the generation of marketing revenue streams as well as market research/statistics in conjunction with client personal profiles.

In VOD, the decoder can be restricted to decode broadcast streams or video only when enabled through supplying a security key. Enabling can be performed, either in real time if connected to the Internet, or at a previous synchronisation of the device, when accessing an Internet authentication/access/billing service provider which provides means for enabling the decoder through authorised payments, or possibly to pay for previously viewed video streams. Similarly to the advertising video streams in the video conferencing, the decoder would log VOD related encoded video streams along with the duration of viewing. This information would be transferred back to the Internet server for market research/feedback and payment purposes.

In the wireless ultrathin client (NetPC) application, real time encoding, transmission and decoding of video streams from Internet or otherwise based computer servers is achieved by adding a unique identifier to the encoded video streams. The client side decoder must be enabled in order to decode the video stream. Enabling of the client side decoder occurs along the lines of the authorised payments in the VOD application or through a secure encryption key process enables various levels of access to wireless NetPC encoded video streams. The computer server encoding software facilitates multiple access levels. In the broadest form, wireless Internet connection includes mechanisms for monitoring client connections through decoder validation fed back from the client decoder software to the computer servers. These computer servers monitor client usage of server application processes and charge accordingly, and also monitor streamed advertising to end clients.

Interactive Audio Visual Markup Language (IAVML)

A powerful component of this system is the ability to control audio-visual scene composition through scripting. With scripts, the only constraints on the composition functions are imposed by the limitations of the scripting language. The scripting language used in this case is IAVML which is derived from the XML standard. IAVML is the textual form for specifying the object control information that is encoded into the compressed bit stream.

IAVML is similar in some respects to HTML, but is specifically designed to be used with object oriented multimedia spatio-temporal spaces such as audio/video. It may be used to define the logical and layout structure of these spaces, including hierarchies, it may also be used to define linking, addressing and also metadata. This is achieved by permitting five basic types of markup tags to provide descriptive and referential information, etc. These are system tags, structural definition tags, presentation formatting, and links and content. Like HTML, IAVML is not case sensitive, and each tag comes in opening and closing forms which are used to enclose the parts of the text being annotated. For example:

<TAG> some text in here </TAG>

Structural definition of audio-visual spaces uses structural tags and include the following:

<SCENE>	Defines video scenes
<STREAMEND>	Demarcate streams within scene
<OBJECT>	Defines object instance
<VIDEODAT>	Defines video object data
<AUDIODAT >	Defines audio object data
<TEXTDAT>	Defines text object data
<GRAFDAT>	Defines vector object data
<VIDEODEFN>	Defines video data format
<AUDIODEFN>	Defines audio data format
<METADATA>	Defines metadata about given object
<DIRECTORY>	Defines directory object
<OBJCONTROL>	Defines object control data
<FRAME>	Defines video frame

The structure defined by these tags in conjunction with the directory and meta data tags permit flexible access to and browsing of the object oriented video bitstreams.

Layout definition of audio-visual objects uses object control based layout tags (rendering parameters) to define the spatio-temporal placement of objects within any given scene and include the following:

<SCALE>	Scale of visual object
<VOLUME>	Volume of audio data
<ROTATION	Orientation of object in 3D space
<POSITION>	Position of object in 3D space

<TRANSPARENT>	Transparency of visual objects
<DEPTH>	Change object Z order
<TIME>	Start time of object in scene
<PATH>	Animation path from start to end time

Presentation definition of audio-visual objects uses presentation tags to define the presentation of objects (format definition) and include the following:

<SCENESIZE>	Scene spatial size
<BACKCOLR>	Scene background colour
<FORECOLR>	Sceneforeground colour
<VIDRATE>	Video Frame rate
<VIDSIZE>	Size of video frame
<AUDRATE>	Audio sample rate
<AUDBPS>	Audio sample size in bits
<TXTFONT>	Text Font type to use
<TXTSIZE>	Text font size to use
<TXTSTYLE>	Text style (bold, underline, italic)

Object Behaviours and action tags encapsulate the object controls and includes the following types:

<JUMPTO>	Replaces current scene or object
<HYPERLINK>	Set hyperlink target
<OTHER>	Retarget control to another object
<PROTECT>	Limit user interaction

<LOOPCTRL>	Looping object control
<ENDLOOP>	Break loop control
<BUTTON>	Define button action
<CLEARWAITING>	Termination waiting actions
<PAUSEPLAY>	Play or pause video
<SNDMUTE>	Mute sound on/off
<SETFLAG>	Set or reset system flag
<SENDFORM>	Send system flags back to server
<CHANNEL>	Change the viewed channel

The hyperlink references within the file permit objects to be clicked on that invoke defined actions.

Simple video menus can be created using multiple media objects with the BUTTON, OTHER and JUMPTO tags defined with the OTHER parameter to indicate the current scene and the JUMPTO parameter indicating the new scene. A persistent menu can be created by defining the OTHER parameter to indicate the background video object and the JUMPTO parameter to indicate the replacement video object. A variety of conditions defined below can be used to customise these menus by disabling or enabling individual options.

Simple forms to register user selections can be created by using a scene that has a number of checkboxes created from 2 frame video objects. For each checkbox object, the JUMPTO and SETFLAG tags are defined. The JUMPTO tag is used to select which frame image is displayed for the object to indicate if the object is selected or not selected, and the indicated system flag registers the state of the selection. A media object defined with

BUTTON and SENDFORM can be used to return the selections to the server for storage or processing.

In cases where there may be multiple channels being broadcast or multicast the CHANNEL tag enables transition between a unicast mode operation to a broadcast or multicast mode and back.

Conditions that may be applied to behaviours and actions (object controls) before they are executed in the client. These are applied in IAVML by creating conditional expressions by using either <IF> or <SWITCH> tags. The client conditions include the following types:

<PLAYING>	Is video currently playing
<PAUSED>	Is video currently paused
<STREAM>	Streaming from remote server
<STORED>	Playing from local storage
<BUFFERED>	Is object frame # buffered
<OVERLAP>	Need to be dragged onto what object
<EVENT>	What user event needs to happen
<WAIT>	Do we wait for conditions to be true
<USERFLAG>	Is the given user flag set?
<AND>	Used to generate expressions
<OR>	Used to generate expressions

Conditions that may be applied at the remote server to control the dynamic media composition process include the following types:

<FORMDATA>	User returned form data

<USERCTRL>	User interaction event has occurred
<TIMEODAY>	Is it a given time
<DAYOFWEEK>	What day of the week is it
<DAYOFYEAR>	Is it a special day
<LOCATION>	Where is the client geographically
<USERTYPE>	Class of user demographic?
<USERAGE>	What is age of user (range)
<USERSEX>	What is the sex of the user (M/F)
<LANGUAGE>	What is the preferred language
<PROFILE>	Other subclasses of user profile data
<WAITEND>	Wait for end of current stream
<AND>	Used to generate expressions
<OR>	Used to generate expressions

An IAVML file will generally have one or more scenes and one script. Each scene is defined to have a determined spatial size, a default background colour and an optional background object in the following manner:

```
<SCENE = "sceneone">
        < SCENESIZE SX = "320",  SY="240">
        < BACKCOLR ="#RRGGBB" >
        <VIDEODAT SRC = "URL">
        <AUDIODAT SRC = "URL">
        <TEXTDAT > this is some text string </a>
</ SCENE>
```

Alternatively, the background object may have been defined previously an then just declared in the scene:

```
<OBJECT = "backgrnd">
        <VIDEODAT SRC = "URL">
        <AUDIODAT SRC = "URL">
        <TEXTDAT > this is some text string </a>
        <SCALE = "2'>
        <ROTATION = "90">
        <POSITION= XPOS ="50" YPOS="100">
</OBJECT>
<SCENE>
        < SCENESIZE SX = "320",  SY="240">
        < BACKCOLR ="#RRGGBB" >
        <OBJECT = "backgrnd">
</SCENE>
```

Scenes can contain any number of foreground objects:

```
<SCENE>
        < SCENESIZE SX = "320",  SY="240">
        < FORECOLR ="#RRGGBB" >
        < OBJECT = "foregnd_object1", PATH ="somepath">
        <OBJECT = "foregnd_object2", PATH ="someotherpath">
        <OBJECT = "foregnd_object3", PATH ="anypath">
</SCENE>
```

Paths are defined for each animated object in a scene:

```
<PATH = somepath>
        < TIME START="0", END="100">
        < POSITION TIME=START, XPOS="0", YPOS="100">
        < POSITION TIME=END, XPOS="0", YPOS="100">
        <INTERPOLATION= LINEAR>
</PATH>
```

Using IAVML, content creators can textually create animation scripts for object oriented video and conditionally define dynamic media composition and rendering parameters. After creation of an IAVML file, the remote server software processes the IAVML script to create the ObjectControl packets that are inserted into the composite video stream that is delivered to the media player. The server also uses the IAVML script internally to know how to respond to dynamic media composition requests mediated by user interaction returned from the client via UserControl packets.

Streaming Error Correction Protocol

In the case of wireless streaming, suitable network protocols must be used to ensure that video data is reliably transmitted across the wireless link to the remote monitor. These may be connection oriented, such as TCP, or connectionless, such as UDP. The nature of the protocol will change, depending on the nature of the wireless network being used, the bandwidth, and the channel characteristics. The protocol must perform the following functions: error control, flow control, packetisation, connection establishment, and link management.

There are many existing protocols for these purposes that have been designed for use with data networks. However, in the case of video, special attention may be required to handle errors, since retransmission of corrupted data is inappropriate due to the real-time constraints imposed by the nature of video on the reception and processing of transmitted data.

To handle this situation the following error control scheme is provided:

(1) Frames of video data are individually sent to the receiver, each with a check sum or cyclic redundancy check appended to enable the receiver to assess if the frame has been received in error;

(2a) If there was no error, then the frame is processed normally;

(2b) If the frame is in error, then the frame is discarded and a status message is sent to the transmitter indicating the number of the video frame that was in error;

(3) Upon receiving such an error status message, the video transmitter stops sending all predicted frames, and instead immediately sends the next available key frame to the receiver;

(4) After sending the key frame, the transmitter resumes sending normal interframe coded video frames until another error status message is received.

A key frame is a video frame that has only been intraframe coded but not interframe coded. Interframe coding is where the prediction processes is performed and makes these frames dependent on all the preceding video frames after and including the last key frame. Key frames are only sent as the first frame and whenever an error occurs. The first frame needs to be a key frame because there is no previous frame to use for interframe coding.

Voice Command Process

Since wireless devices are small, the ability to enter text commands manually for operating the device and data processing is difficult. Voice commands have been suggested as a possible avenue for achieving hands-free operation of the device. This presents a problem in that many wireless devices have very low processing power, well below that required for general automatic speech recognition (ASR). The solution in this case is to capture the user speech on the device, compress it, and send it to the server for ASR and execution, since in any case the server will be actioning all user commands. This frees the device from having to perform this complex processing, since it is likely to be devoting most of its processing resources to decoding and rendering any streaming audio/video content.

Applications

Ultrathin Client Process and Compute Servers

By using an ultra thin client as a means for controlling a remote computer of any kind from any other kind of personal mobile computing device, a virtual computing network is created. In this new application, the user's computing device performs no data processing, but only serves as a user interface into the virtual computing network. All the data processing is performed by compute servers located in the network. At most, the terminal is limited to decoding all output and encoding all input data, including the actual user interface display. Architecturally, the incoming and outgoing data streams are totally independent within the user terminal. Control over the output or displayed data is performed at the compute server where the input is data is processed. Accordingly, the graphical user interface (GUI) decomposes into two separate data streams: the input and the output display component, which is a video. The input stream is a command sequence that may be a combination of ASCII characters together with mouse or pen events. To a large extent, decoding and rendering the display data comprises the main function of such a terminal and complex GUI displays can be rendered.

Figure 18 shows an ultra thin client system operating in a wireless LAN environment. This system may have a range from 300 meters indoors to up to 1 km outdoors. The ultrathin client is a personal digital assistant or palmtop computer with a wireless network card and antenna to receive signals. The wireless network card interfaces to the personal digital assistant through a PCMCIA slot or a compact flash port. The compute server may be any computer running a GUI that is connected to the internet or a local area network with wireless LAN capability. The computer server uses the video encoder to convert the GUI display and any audio to compressed video using the process described previously and transmits it to the ultra thin client. The GUI display may be captured using a GUI screen reading means which is a standard function in many operating systems such as CopyScreenToDIB() in Microsoft Windows NT. The ultra thin client receives the compressed video and renders it appropriately to the user display using the video decoder. Any user control data is transmitted back to the compute server, where it is interpreted and used to control the compute server. This includes the ability to execute new programs, terminate programs, perform operating system functions, and any other functions associated with the running program(s). This control may be effected through various means, in the case of MS Windows NT, the Hooks/JournalPlaybackFunc() can be used.

For longer range applications, the WAN system of Figure 19 is preferred. In this case, the compute server is directly connected to a standard telephone interface for transmitting the signals across a CDMA or GSM cellular phone network. The ultra thin client in this case comprises a personal digital assistant with a modem connected to a phone. All other aspects are the same. In a variation of this system, the PDA and phone are integrated within a single device. In one instance of this ultra thin client system, the mobile device has full access to the compute server from any location whilst within the reach of standard mobile telephony networks such as CDMA or GSM. A cabled version of this system may also be used which dispenses with the mobile phone so that the ultra thin computing device is connected directly to the standard cabled telephone network through a modem.

The compute server may also be remotely located and connected via the Internet to a local wireless transmitter/receiver as depicted in Figure 20. This ultra thin client application is especially relevant in the context of emerging Internet-based virtual computing systems.

Rich Audio-Visual User Interfaces

In the ultra thin client system where no object control data is inserted into the bit stream, the client performs no process other than rendering a single video object to the display and returns all user interaction to the server for processing. While that approach can be used to access the graphical user interface of remotely executing processes, it is not suitable for creating user interfaces for locally executing processes.

Given the object-based capabilities of the DMC and interaction engine, this overall system and its client-server model is particularly suited for use as the core of a rich audio-visual user interface. Unlike typical graphical user interfaces which are based on the concept of mostly static icons and rectangular windows, the current system is capable of creating rich user interfaces using multiple video and other media objects.

Multipart Wireless VideoConferencing Process

Figure 21 shows a multiparty wireless videoconferencing system involving two or more wireless client telephony devices. In this application, two or more participants may set up a number of video communication links among themselves. There is no centralised control mechanism, and each participant may decide what links to activate in a multiparty conference. For example, in a three person conference consisting of persons A,B,C, links may be formed between persons AB, BC and AC (3 links), or alternatively AB and BC but not AC (2 links). In this system, each user may set up as many simultaneous links to different participants as they like, as no central network control is required and each link is separately managed. The incoming video data for each new videoconference link forms a new video object stream that is fed into the object oriented video decoder of each wireless device. In this application, the object video decoder is run in a presentation mode where each video object is rendered according to layout rules, based on the number of video objects being displayed. One of the video objects is identified as currently active, and this one is rendered in a larger size than the other objects. The selection of which object is currently active may be performed using either automatic means based on the video object with most acoustic energy (loudness/time) or manually by the user. Client telephony devices include personal digital assistants and handheld personal computers with wireless network cards and antennae to receive and transmit signals. A wireless network card interfaces to the client telephony device through a PCMCIA slot, a compact flash port or other means. Each client telephony device may include a video camera for digital video capture and one or more microphones for audio capture. The client telephony device includes the video encoder to compress the captured video and audio signals, using the process described previously, which are then transmitted to one or more other client telephony devices. The digital video camera may only capture digital video and pass it to the client telephony device for compression and transmission, or it may also compress the video itself using a VLSI hardware chip (an ASIC) and pass the coded video to the telephony device for transmission. The client telephony devices, which contain specific software, receive the compressed video and audio signals and render them appropriately to the user display and speaker outputs using the process previously described. This embodiment may also include direct video manipulation or advertising on a client telephony device, using the process of interactive object manipulation described previously, which can be reflected through the same means as above to other client telephony devices participating in the same videoconference. This embodiment may include transmission of user control data between client telephony devices such as to provide a means for remote control of other client telephony devices. Any user control data is transmitted back to the appropriate client telephony device, where it is interpreted and then used to control local video image and other software and hardware functions. As in the case of the ultra thin client system application, there are various network interfaces which can be used.

Interactive Animation or Video On Demand with Targeted Instream User Advertising

Figure 22 is a block diagram of an interactive video on demand system with targeted user video advertising. In this system, a news or video-on-demand (VOD) provider would unicast or multicast video data streams to individual subscribers. In one instance of the video decoder, a small video advertisement object is dynamically composed into the video stream being delivered to the decoder and to be rendered into the scene being viewed at certain times. This video advertising object can be changed either from pre-downloaded advertising stored on the device in a library, or preferably streamed from an online video server capable of dynamic media composition - targeted specifically to the client device based on the client owner's server stored profile. For targeted video based advertising, feedback and control mechanisms for video streams and viewing thereof are used. The VOD provider maintains and operates a video server that stores compressed video streams. When a subscriber selects a program from the video server, the provider's transmission system automatically selects what promotion or advertising data is applicable from information obtained from a subscriber profile database that includes information such as subscriber age, gender geographical location, subscription history, etc. The advertising data, which is stored as a single video object, is then inserted into the transmission data stream together with the requested video data and sent to the user. As a separate video object, the user can then interact with the advertising video object by adjusting its presentation/display properties. The user may also interact with the advertising video objects by clicking on the object to thereby send a message back to the video server indicating that the user wishes to activate some function associated with that advertising video object as determined by the VOD provider. This function may simply entail a request for further information from the advertiser, placing a video/phone call to the advertiser, or some other form of control. In addition to advertising, this function may be directly used by the VOD provider to promote additional video offerings such as other available channels, that may be advertised as small moving iconic images. In this case, the user action of clicking on such an icon may be used by the provider to change the primary video data being sent to the subscriber or send additional data. Multiple video object data streams may be combined by the video object overlay means into the final composite video data stream that is transmitted to each client. Each of the separate video object streams that are combined may be retrieved over the Internet by the video promotion selection means from different remote sources such as other video servers, web cameras, or compute servers. Again, as in the other system applications of ultra thin clients and videoconferencing, various preferred network interfaces can be used.

In one embodiment of instream advertising, the video advertisement object may be programmed to operate like a button which, when selected by a user, may do one of the the following:

- Immediately change the video scene being viewed by jumping to a new scene that provides more information about the product being advertised or to an online e-commerce enabled store. For example, it may be used to change "video channels".

- Immediately change the video advertising object into streaming text information like subtitles by replacing the object with another that provides more information about the product being advertised. This does not affect any other video objects in the displayed scene.
- Removes the video advertising object and sets a system flag indicating that the user has selected the advertisement, the current video will then play through to the end normally and then jumpto the indicated advertisement target
- Send a message back to the server registering interest in the product being offered for future asynchronous followup information, which may be via email or as additional streaming video objects, etc.
- Where the video advertising object is being used for branding purposes only, clicking on the object may toggle its opacity and make it semitransparent, or enable it to perform a predefined animation such as rotating in 3D or moving in a circular path.

In another embodiment, the dynamic media composition capabilities of this video system may be used to enable viewers to customise their content. An example is where the user may be able to select from one of a number of characters to be the principal hero. In a typical case with an animated cartoon, viewers may be able to select from male or female characters. This selection may be performed interactively or may be based on a stored user profile. Selecting a male character would cause the male character's audiovisual media object to be composited into the bit stream to replace that of a female character. In another example, rather than just selecting the principal character for a fixed plot, the plot itself may be changed by making selections during viewing that change the storyline by selecting which scene to jumpto next. A number of alternative scenes would be available at each point, but the selection could be constrained by the previous selections and where in the storyline the video is at.

Internet service providers may provide user authentication and access control to video material, metering of content consumption and billing of usage. In this situation, all users must register with the relevant authentication/access provider before they can use the system. The authentication/access service creates a unique identifier for each user and that is automatically stored by the client system. All subsequent requests to video content providers by users must be performed with the use of a valid user identifier. The content providers, as shown in Figure 23, then liaise with the billing service which provides a one-time access key to the user to enable the user to view the video provided by the content provider. The access control and or billing service provider keep a user usage profile which may then be sold or licensed to third parties for advertising/promotional purposes. In order to implement billing and usage control, a suitable encryption method must be deployed, as previously described. In addition to this, a process for uniquely branding/identifying an encoded video is required as described previously.

Video Advertising Brochures

An interactive video file may be downloaded rather than streamed to a device so that it can be viewed offline at any time. A downloaded video file still preserves all of the interaction and dynamic media composition capabilities that are provided by the streaming case. Hence video brochures may include menus, advertising objects, and even forms that register user selections and feedback. The only difference is that, since it is offline, the video can not be made to jump to a new URL which is not located on the device. In this situation, the player must store all user selections and forward these to the appropriate remote server the next time the device is online or synchronised with a PC.

Distribution Models and DMC Operation

The on demand streaming embodiment provides a channel with low bandwidth and low latency, while the FTP-based download embodiment typically provides a high bandwidth, high latency channel. Users can view remotely stored video and animation content either on demand by viewing it as it is streamed in real time from the server to the client, or by immediately downloading the entire video first at high speed and then watching it. In another embodiment that provides low bandwidth and high latency, the device is said to be "always on". This means that the client device is always online. In this case, the video content can be trickled down to the device overnight or other off-peak period and buffered in memory for viewing at a later time. In this model, the operation of the system differs from the download case in that a user's request is registered at the server. The server then waits until the requested data can either be delivered during an off-peak period of network utilisation and then sends the data to the client, or alternatively sends the data in small amounts from time-to-time using any available residual bandwidth left over in the network from that allocated to constant rate connections. In either case, as the user would be unaware that the requested data has been fully delivered, the fact must be signalled to users so that they can then view the requested data when they are ready.

These three distribution models are suitable for unicast mode of operation. In the on demand processing model, the remote streaming server can perform unrestricted dynamic media composition and handle user interaction, etc, in real-time, whereas in the other two models, the local client must handle the user interaction and perform DMC as the user will view the content offline. Any user data interaction data and form data to be returned to the server will be delivered at some future, indeterminate time and will be processed offline.

Apart from unicast, other operating modes include multicast and broadcast. In the case of a multicast or broadcast, the system/user interaction and DMC capabilities are constrained and must operate in a different manner. In a wireless environment, it is likely that multicast and broadcast data will be transmitted in separate channels. These are not purely logical channels as with packet networks, instead these may be circuit switched channels. A single transmission is sent from one server to multiple clients. Hence user interaction data may be returned to the server using separate individual unicast 'back channel' connections for each user. The distinction between multicast and broadcast is that multicast data is broadcast only within certain geographical boundaries such as the range of a radio cell. In the broadcast mode the data is sent to all radio cells within a network.

An example of how a broadcast channel may be used is to transmit a cycle of scenes containing service directories. Scenes could be categorised to contain a set of hyper-linked video objects corresponding to other selected broadcast channels, so that users selecting an object will change to the relevant channel. Another scene may contain a set of hyper-linked video objects pertaining to video-on-demand services, where the user, by selecting a video object, would create a new unicast channel and switch from the broadcast to that. Similarly, hyper-linked objects in a unicast on demand channel would be able to change the bit stream being received by the client to that from a specified broadcast channel

Since a multi or broadcast channel will convey the same data from the server to all the clients, the DMC is restricted in its ability to customise the scene for each user. The control of the DMC for the channel is not subject to individual users, so it is not possible for user interaction to modify the content of the bit stream being broadcast. Since broadcast relies on real-time streaming, it is not possible to use the same approach for local client DMC with offline viewing, where each scene can have multiple object streams. The user, however, is not completely inhibited from interacting with the scenes, they are still free to modify rendering parameters such as activating animations, etc, registering object selection with the server, and they are free to select a new unicast or broadcast channel to jump to by activating any hyperlinks associated with video objects.

One way in which DMC can be used to customise the user experience in broadcast is to monitor the distribution of different users currently watching the channel and construct the outgoing bit stream defining the scene to be rendered based on the average user profile, For example, the selection of instream advertising may be based on whether viewers were predominantly male or female. Another manner that the DMC can be used to customise the user experience in a broadcast situation is to send a composite bit stream with multiple media objects, without regard for the current viewer distribution. The client in this case must select from among the objects based on a user profile local to the client to create the final scene. For example, multiple subtitles in a number of languages may be inserted into the bit stream defining a scene for broadcasting. The client is then able to select which language subtitle to render based on special conditions in the object control data broadcast in the bit stream.

Video Surveillance System

Figure 24 is a block diagram of a video security/surveillance system. In addition to transmitting remote video to wireless handheld devices over short ranges using wireless LAN interfaces, security devices are also able to transmit remote video over long distances using a standard telephone interface over a CDMA or GSM system. Other access network architectures can also be used. The security system can have intelligent functions such as motion detection alarms, automatic notification and dial out on alarm, recording and retrieval of video segments, select and switch between multiple camera inputs, and provide for user activation of multiple digital or analogue outputs at the remote location.

Electronic Greeting Card Service

Figure 25 is a block diagram of an electronic greeting card service for smart mobile phones. In this system, a user can access a greeting card server either from the Internet or the mobile phone network. The server provides a software interface that permits users to customise a greeting card template selected from a library stored on the server. The templates are short videos or animations covering a number of themes, such as birthday wishes, postcards, good luck wishes, etc. The customisation may include the insertion of text and or audio content to the video and animation templates. After customisation, the user may pay for the transaction and forward the electronic greeting card to a person's mobile phone number. The electronic greeting will then be stored by the server and passed into the wireless phone network during off-peak periods. In the case of post cards, specialised template videos can be created for mobile phone networks in each geographic locations that can only be sent by people physically within that locality.

Wireless local loop streaming video and animation system

Another application is for wireless access to corporate audio-visual training materials stored on a local server, or for wireless access to audio-visual entertainment such as music videos in domestic environments. The main problem encountered in wireless streaming is the low bandwidth capacity of wide area wireless networks and high costs. Hence streaming high quality video requires high link bandwidth, and is not currently possible over wireless networks. An alternate solution is to spool the video to be viewed over a typical low speed wide area network connection to a local wireless server and, once this has been fully received, it is available for wireless transmission over a high capacity local loop or private wireless network.

Perhaps the largest application for this is local wireless streaming of music videos. A user downloads a music video from the Internet onto a local computer attached to a wireless domestic network. These music videos can then be streamed to a PDA or pocket or wearable computing device that also has wireless connectivity. A software management system running on the local computer server manages the library of videos, and responds to client user commands from the PDA to control the streaming process.

There are four main components to the server side software management system: a browsing structure creation component; a user interface component; a streaming control component; and a network protocol component. The browsing structure creation component creates the data structures that are required to create a user interface for browsing locally stored videos. In one embodiment, the user may create a number of playlists using the server software; these playlists are then formatted by the user interface component for transmission to the client player. Alternatively, the user may store the video data in a hierarchical file directory structure and the browsing structure component creates the browsing data structure by automatically navigating the directory structure. The user interface component formats browsing data for transmission to the client and receives commands from the client that are relayed to the streaming control component. The user play back controls may include 'standard' functions such as play start, pause stop, loop etc. In one embodiment, the user interface component formats the browsing data into HTML, but the user playback controls into a custom format. In this embodiment, the client user interface consists of a two separate components: a HTML browser handles the browsing functions, while the playback control functions are handled by the video decoder/player. In another embodiment, there is no separation of function in the client software, and the video decoder/player handles all of the user interface functionality itself. In this case, the user interface component formats the browsing data into a custom format understood directly by the video decoder/player.

This application is most suitable for implementation in domestic or corporate applications, for training or entertainment purposes. For example, a technician may use the configuration to obtain audio-visual training materials on how to repair or adjust a faulty device without having to move away from the work area to a computer console in a separate room. Another application is for domestic users to view high quality audio-visual entertainment while lounging outside in their patio. The back channel allows user to select what audio video content they wish to view from a library. The primary advantage is that the video monitor is portable and therefore the user can move freely around the office or home. It will be appreciated that this is a significant improvement over known prior art of electronic books and streaming over wireless cellular networks.

Object Oriented Data Format

The object oriented multimedia file format is designed to meet the following goals:

- **Speed** – the files are designed to be rendered at high speed
- **Simplicity** – the format is simple so that parsing is fast and porting is easy. In addition, compositing can be performed by simply appending files together.
- **Extensibility** – The format is a tagged format, so that new packet types can be defined as the players evolve, while maintaining backwards compatibility with older players.
- **Flexibility** – There is a separation of data from its rendering definitions, permitting total flexibility such as changing data rates, and codecs midstream on the fly.

The files are stored in big-endian byte order. The following data types are used:

Type	Definition
BYTE	8 bits, unsigned char
WORD	16 bits, unsigned short
DWORD	32 bits, unsigned long
BYTE[]	String, byte[0] specifies length up to 254, (255

	reserved)
IPOINT	12bits unsigned, 12 bits unsigned, (x,y)
DPOINT	8 bits unsigned char, 8 bits unsigned char, (dx,dy)

The file stream is divided into packets or blocks of data. Each packet is encapsulated within a container similar to the concept of atoms in Quicktime, but is not hierarchical. A container consists of a **BaseHeader** record that specifies the payload type and some auxiliary packet control information and the size of the data payload. The payload type defines the various kinds of packet in the stream. The one exception to this rule is the SystemControl packet used to perform end-to-end network link management. These packets only consist of a BaseHeader with no payload. In this case, the payload size field is reinterpreted. In the case of streaming over circuit switched networks, a preliminary, additional network container is required to achieve error resilience by providing for synchronisation and checksums

There are four main types of packets within the bit stream: data packets, definition packets, control packets and metadata packets of various kinds. Definition packets are used to convey media format and codec information that is used to interpret the data packets. Data packets convey the compressed data to be decoded by the selected application. Hence an appropriate Definition packet must precede any data packets of for each given data type. Control packets that define rendering and animation parameters must occur after Definition but before Data Packets.

Conceptually, the object oriented data can be considered to consist of 3 main interleaved streams of data. The definition, data, control streams. The metadata is an optional fourth stream. These 3 main streams interact to generate the final audio-visual experience that is presented to a viewer.

All files start with a SceneDefinition block which defines the AV scene space into which any audio or video streams or objects will be rendered. Metadata and directory packets contain additional information about the data contained by the data and definition packets to assist browsing of the data packets. If any metadata blocks exist, they must occur immediately after a SceneDefinition packet. A directory packet must immediately follow a Metadata packet or a SceneDefinition packet if there is no Metadata packet.

The file format permits integration of diverse media types to support object oriented interaction, both when streaming the data from a remote server or accessing locally stored content. To this end, multiple scenes can be defined and each may contain up to 200 separate media objects simultaneously. These objects may be of a single media type such as video, audio, text or vector graphics, or composites created from combinations of these media types.

As shown in Figure 4, the file structure defines a hierarchy of entities: a file can contain one of more scenes, each scene may contain one of more objects, and each object can contain one or more frames. In essence, each scene consists of a number of separate interleaved data streams, one for each object each consisting of a number of frames. Each stream is consists of one of more definition packets, followed by data and control packets all bearing the same object_id number.

Stream Syntax

Valid Packet Types

The BaseHeader allows for a total of up to 255 different packet types according to payload. This section defines the packet formats for the valid packet types as listed in the following table.

Value	DataType	Payload	Comment
0	SCENEDEFN	SceneDefinition	Defines scene space properties
1	VIDEODEFN	VideoDefinition	Defines video format / codec properties
2	AUDIODEFN	AudioDefinition	Defines audio format / codec properties
3	TEXTDEFN	TextDefinition	Defines text format / codec properties
4	GRAFDEFN	GrafDefinition	Defines vector graphics format / codec properties
5	VIDEOKEY	VideoKey	Video Key Frame data
6	VIDEODAT	VideoData	Compressed Video data
7	AUDIODAT	AudioData	Compressed audio data
8	TEXTDAT	TextData	Text data
9	GRAFDAT	GrafData	Vector Graphics data
10	FLASHDAT	FlashData	Flash animation in Shock Wave Format
11	OBJCTRL	ObjectControl	Defines object animation / rendering properties
12	SYSCTRL	-	Used for streaming end to end link management
13	USERCTRL	UserControl	Back channel for user system interaction
14	METADATA	MetaData	Contains meta data about AV scene
15	DIRECTORY	Directory	Directory of data or system objects,
16	VIDEOENH	-	RESERVED – video enhancement data

17	AUDIOENH	-	RESERVED – audio enhancement data
18	VIDEOEXTN	-	RESERVED - Redundant I frames
19	VIDEOTERP	Video Data	Discardable Interpolated video files
255	-	-	RESERVED

BaseHeader

Short BaseHeader is for packets that are shorter than 65536 bytes

Description	Type	Comment
Type	BYTE	Payload packet type [0], can be definition, data or control packet
Obj_id	BYTE	Object stream ID – what object does this belong to
Seq_no	WORD	Frame sequence number, individual sequence for each object
Length	WORD	Size of frame to follow in bytes {0 means end of stream}

Long BaseHeader will support packets from 64K up to 0xFFFFFFFF bytes

Description	Type	Comment
Type	BYTE	Payload packet type [0], can be definition, data or control packet
Obj_id	BYTE	Object stream ID – what object does this belong to
Seq_no	WORD	Frame sequence number, individual sequence for each object
Flag	WORD	0xFFFF
Length	DWORD	Size of frame to follow in bytes – 0xFFFF

System BaseHeader is for end-to-end network link management

Description	Type	Comment

Type	BYTE	DataType = SYSCTRL
Obj_id	BYTE	Object stream ID – what object does this belong to
Seq_no	WORD	Frame sequence number, individual sequence for each object
Status	WORD	StatusType {ACK, NAK, CONNECT, DISCONNECT, IDLE}

Total size is 6 or 10 bytes

SceneDefinition

Description	Type	Comment
Magic	BYTE[4]	ASKY = 0x41534B59 (used for format validation)
Version	BYTE	Version 0x00- current
Compatible	BYTE	Version 0x00- current - minimum format playable
Width	WORD	SceneSpace width (0 = unspecified)
Height	WORD	SceneSpace height (0 = unspecified)
~~BackFill~~	~~WORD~~	~~RESERVED – Scene Fill Style / colour~~
NumObjs	BYTE	How many objects in this scene
Mode	BYTE	Frame playout mode bitfield

Total size is 14 bytes

MetaData

Description	Type	Comment
NumItem	WORD	Number of scenes/frames in file/scene (0 = unspecified)
SceneSize	DWORD	Size in bytes of file/scene/object including (0 = unspecified)
SceneTime	WORD	Playing time of file/scene/object in seconds (0 = unspecified/static)
BitRate	WORD	Bit rate of this file/scene/object in kbits /sec

MetaMask	DWORD	Bit field specifying what optional 32 meta data tags follow.
Title	BYTE[]	Title of video file/scene - whatever you like, byte[0] = length
Creator	BYTE[]	Who created this, byte[0] = length
Date	BYTE[8]	Creation date in ASCII => DDMMYYYY
Copyright	BYTE[]	
Rating	BYTE	X,XX,XXX etc
EncoderID	BYTE[]	-
-	BYTE	-

Directory
This is an array of type WORD or DWORD. The size is given by the Length field in the BaseHeader packet.

VideoDefinition

Description	Type	Comment
Codec	BYTE	Video Codec Type { RAW, QTREE };
Frate	BYTE	Frame rate {0 = stop/pause video play} in 1/5 sec
Width	WORD	Width Of video frame
Height	WORD	Height Of video frame
Time	DWORD	Time stamp in 50ms resolution from start of scene (0 = unspecified)

Total size is 10 bytes

AudioDefinition

Description	Type	Comment
Codec	BYTE	Audio Codec Type { RAW, G723, , ADPCM }
Format	BYTE	Audio Format in bits 7-4, Sample Rate in bits 3-0
Fsize	WORD	Samples per frame
Time	DWORD	Time stamp in 50ms resolution from start of scene (0 = unspecified)

Total size is 8 bytes

5 **TextDefinition**

Description	Type	Comment
Type	BYTE	Type in low nibble {TEXT, HTML, etc} compression in high nibble
Fontinfo	BYTE	Font size in low nibble, Front Style in high nibble
Colour	WORD	Font colour
BackFill	WORD	Background colour
Bounds	WORD	Text boundary Box (frame) X in high byte, Y in low byte
Xpos	WORD	Xpos relative to object origin if defined relative to 0,0 otherwise
Ypos	WORD	Xpos relative to object origin if defined relative to 0,0 otherwise
Time	DWORD	Time stamp in 50ms resolution from start of scene (0 = unspecified)

Total size is 16 bytes

GrafDefinition

Description	Type	Comment
Xpos	WORD	XPos relative to object origin if defined relative to 0,0 otherwise
Ypos	WORD	XPos relative to object origin if defined relative to 0,0 otherwise
FrameRate	WORD	Frame delay in 8.8 fps
FrameSize	WORD	RESERVED Frame size in twips (1/20 pel)– used for scaling to fit scene space
Time	DWORD	Time stamp in 50ms resolution from start of scene

Total size is 12 bytes

VideoKey, VideoData, AudioData, TextData, GrafData and FlashData

Description	Type	Comment
Payload	-	Compressed data

UserControl

Description	Type	Comment
Event	BYTE	User data Type eg. PENDOWN, KEYEVENT, PLAYCTRL,
Key	BYTE	Parameter 1 = Keycode value / Start / Stop / Pause
HiWord	WORD	Parameter 2 =X position
LoWord	WORD	Parameter 3 =Y position
Time	WORD	Timestamp = sequence number of activated object
Data	BYTE[]*	Optional field for form field data

Total size is 8+ bytes

ObjectControl

Description	Type	Comment
Actions	WORD	Bit field actions defined in remaining payload
Params	...	Parameters for actions defined by Action bit field

Semantics

BaseHeader

This is the container for all information packets in the stream.

Type - BYTE

Description – Specifies the type of payload in packet as defined above

Valid Values: enumerated 0 –255, see Payload type table below

Obj_id - BYTE

Description – Object ID – defines scope - what object does this packet belong to. Also defines the Z-order in steps of 255, that increases towards the viewer. Up to four different media types can share the same obj_id.

Valid Values: 0 – NumObjs (max 200) NumObjs defined in SceneDefinition
201-253: Reserved for system use
254: Directory of Scenes
255: This File

Seq_no - WORD

Description – Frame sequence number, individual sequence for each media type within an object. Sequence number are restarted after each new SceneDefinition packet.

Valid Values: 0 – 0xFFFF

Flag (optional) - WORD

Description – Used to indicate long baseheader packet.

Valid Values: 0xFFFF only

Length – WORD / DWORD

Used to indicate payload length in bytes, (if flag set packet size = length + 0xFFFF).

Valid Values: 0x0001 - 0xFFF, If flag is set 0x00000001 – 0xFFFFFFFF ()

0 - RESERVED for Endof File / Stream 0xFFFF only

Status- WORD

Used only with SysControl DataType flag, for end to end link management.

Valid Values: enumerated 0 – 65535

Value	Type	Comment
0	ACK	Acknowledge packet with given obj_id and seq_no
1	NAK	Flag error on packet with given obj_id and seq_no
2	CONNECT	Establish client / server connection
3	DISCONNECT	Break client / server connection
4	IDLE	Link is idle
5-65535	-	RESERVED

SceneDefinition

This defines the properties of the AV scene space into which the video and audio objects will be played.

Magic – BYTE[4]

Description – used for format validation,

Valid Value: ASKY = 0x41534B59

Version – BYTE
Description – used for stream format validation
Valid Range: 0 - 255 (current = 0)

Compatible – BYTE
Description – what is the minimum player that can read this format
Valid Range: 0 - Version

Width – WORD
Description – SceneSpace width in pixels
Valid Range: 0x0000 – 0xFFFF

Height – WORD
Description – SceneSpace height in pixels
Valid Range: 0x0000 – 0xFFFF

BackFill – (RESERVED) WORD
Description –background scene fill (bitmap, solid colour, gradient)
Valid Range: 0x1000 – 0xFFFF solid colour in 15 bit format
else the low order BYTE defines the object id for a vector object and the high order BYTE (0 – 15) is an index to gradient fill style table
This vector object definition must occur prior to any data control packets

NumObjs – BYTE
Description – how many data objects are in this scene
Valid Range: 0 – 200 (201-255 reserved for system objects)

Mode – BYTE

Description - Frame playout mode bitfield

Bit: [7] play status - paused = 1, play = 0 // continuous play or step through

Bit: [6] Zooming - prefer = 1, normal = 0 // play zoomed

Bit: [5] RESERVED - data storage - live = 1, stored = 0 // is this being streamed ?

Bit: [4] RESERVED streaming - reliable = 1, best try = 0 // is streaming reliable

Bit: [3] RESERVED data source - video = 1, thinclient = 0 // originating source

Bit: [2] RESERVED Interaction - allow = 1, disallow = 0

Bit: [1] RESERVED

Bit: [0] RESERVED – linear = 1, Nonlinear = 0 // play scene linearly

MetaData

This specifies metadata associated with either an entire file, scene or an individual AV object. Since files can be concatenated, there is no guarantee that a metadata block with file scope is valid past the last scene it specifies. Simply comparing the file size with the SCENESIZE field in this Metadata packet however can validate this.

The OBJ_ID field in baseHeader defines the scope of a metadata packet. This scope can be the entire file (255), a single scene (254), or an individual video object (0-200). Hence if MetaData packets are present in a file they occur in flocks (packs?) immediately following SceneDefinition packets.

NumItem – WORD

Description – Number of scenes/frames in file/scene, For scene scope NumItem contains the number of frames for video object with obj_id=0

Valid Range: 0-65535 (0 = unspecified)

SceneSize – DWORD
Description – Self inclusive size in bytes of file/scene/object including,
Valid Range: 0x0000-0xFFFFFFFF (0 = unspecified)

SceneTime – WORD
Description – Playing time of file/scene/object in seconds,
Valid Range: 0x0000-0xFFFF (0 = unspecified)

BitRate – WORD
Description – bit rate of this file/scene/object in kbits /sec,
Valid Range: 0x0000-0xFFFF (0 = unspecified)

MetaMask – (RESERVED) DWORD
Description – Bit field specifying what optional 32 meta data fields follow in order,
Bit Value [31]: Title
Bit Value [30]: Creator
Bit Value [29]: Creation Date
Bit Value [28]: Copyright
Bit Value [27]: Rating
Bit Value [26]: EncoderID
Bit Value [26-27]: RESERVED

Title – (Optional) BYTE[]
Description – String of up to 254 chars

Creator – (Optional) BYTE[]
Description – String of up to 254 chars

Date – (Optional) BYTE[8]
Description – Creation date in ASCII => DDMMYYYY

Copyright – (Optional) BYTE[]
Description – String of up to 254 chars

Rating – (Optional) BYTE
Description – BYTE specifying 0-255

Directory

This specifies directory information for an entire file or for a scene. Since the files can be concatenated, there is no guarantee that a metadata block with file scope is valid past the last scene it specifies. Simply comparing the file size with the SCENESIZE field in a Metadata packet however can validate this.

The OBJ_ID field in baseHeader defines the scope of a directory packet. If the value of the OBJ_ID field is less than 200 then the directory is a listing of sequence numbers (WORD) for keyframes in a video data object. Else, the directory is a location table of system objects. In this case the table entries are relative offset in bytes (DWORD) from the start of the file (for directories of scenes and directories) or scene for other system objects). The number of entries in the table and the table size can be calculated from the LENGTH field in the BaseHeader packet.

Similar to MetaData packets if Directory packets are present in a file they occur in flocks (packs?) immediately following SceneDefinition, or Metadata packets.

VideoDefinition

Codec – BYTE

Description – Compression Type

Valid Values: enumerated 0-255

Value	Codec	Comment
0	RAW	Uncompressed, the first byte defines colour depth
1	QTREE	Default Video codec
2-255	-	RESERVED

Frate – BYTE

Description – frame playout rate in 1/5 sec (ie max = 51 fps, min = 0.2 fps)

Valid Values: 1 – 255, play / start playing if stopped

0 – stop playing

Width – WORD

Description – how wide in pixels in video frame

Valid Values: 0 - 65535

Height – WORD

Description – how high in pixels in video frame

Valid Values: 0 - 65535

Times – WORD

Description – Time stamp in 50ms resolution from start of scene (0 = unspecified)

Valid Values: 1 – 0xFFFFFFFF (0 = unspecified)

AudioDefinition

Codec – BYTE

Description – Compression Type

Valid Values: enumerated 1 (0 = unspecified)

Value	Codec	Comment
0	WAV	Uncompressed
1	G723	Default Video codec
2-255	-	RESERVED

Format – BYTE

Description – This BYTE is split into 2 separate fields that are independently defined. The top 4 bits define the audio format (Format >> 4) while the bottom 4 bits separate define the sample rate (Format & 0x0F).

Low 4 Bits, Value: enumerated 0 – 15, Sampling Rate

Value	Samp.Rate	Comment
0	0	0 – stop playing
1	5.5 kHz	5.5 kHz Very low rate sampling, start playing if stopped
2	8 kHz	Standard 8000 Hz Sampling, start playing if stopped
3	11 kHz	Standard 11025 Hz Sampling, start playing if stopped
4	16 kHz	2x 8000 Hz Sampling, start playing if stopped
5	22 kHz	Standard 22050 Hz Sampling, start playing if stopped
6	32 kHz	4x 8000 Hz Sampling, start playing if stopped
7	44 kHz	Standard 44100 Hz Sampling, start playing if stopped
8-15		RESERVED

High 4 Bit, Value: enumerated 0-15, Format

Value	Codec	Comment
0	MONO8	Monophonic, 8 bits per sample
1	MONO16	Monophonic, 16 bits per sample
2	STEREO8	Stereophonic, 8 bits per sample
3	STEREO16	Stereophonic, 16 bits per sample
4	JOINT8	RESERVED
5	JOINT16	RESERVED
6	DUAL8	RESERVED
7	DUAL16	RESERVED

8-15		RESERVED

Fsize – WORD

Description – samples per frame

Valid Values: 0 - 65535

Times – WORD

Description – Time stamp in 50ms resolution from start of scene (0 = unspecified)

Valid Values: 1 – 0xFFFFFFFF (0 = unspecified)

TextDefinition

Type – BYTE

Description – Defines how text data is interpreted in low nibble (Type & 0x0F) and compression method in high nibble (Type >> 4)

Low 4 Bits, Value: enumerated 0 – 15, Type - interpretation

Value	Type	Comment
0	PLAIN	Plain text – no interpretation
1	TABLE	RESERVED – table data
2	FORM	Form / Text Field for user input
3	WML	RESERVED WAP - WML
4	HTML	RESERVED HTML
5-15	-	RESERVED

High 4 Bit, Value: enumerated 0-15, compression method

Value	Codec	Comment
0	NONE	Uncompressed 8 bit ASCII codes

1	TEXT7	RESERVED – 7 Bit Character codes
2	HUFF4	RESERVED - 4 bit Huffman coded ASCII
3	HUFF8	RESERVED - 8 bit Huffman coded ASCII
4	LZW	RESERVED - Lepel-Zev-Welsh coded ASCII
5	ARITH	RESERVED – Arithmetic coded ASCII
6-15	-	RESERVED

FontInfo – BYTE

Description – Size in low nibble (FontInfo & 0x0F) Style in high nibble (FontInfo >>4). This field is ignored if the Type is WML or HTML.

Low 4 Bits Value: 0 – 15 FontSize

High 4 Bit Values: enumerated 0-15, FontSyle

Colour – WORD

Description – Textface colour

Valid Values: 0x0000 – 0xEFFF, colour in 15 bit RGB (R5,G5,B5)
0x8000 – 0x80FF, colour as index into VideoData LUT (0x80FF = transparent)
0x8100 – 0xFFFF RESERVED

BackFill – WORD

Description – Background colour

Valid Values: 0x0000 – 0xEFFF, colour in 15 bit RGB (R5,G5,B5)
0x8000 – 0x80FF, colour as index into VideoData LUT (0x80FF = transparent)
0x8100 – 0xFFFF RESERVED

Bounds – WORD

Description – Text boundary box (frame) in character units, Width in the LoByte (Bounds & 0x0F) and height in the HiByte (Bounds >> 4). The text will be wrapped using the width and clipped for the height.

Valid Values: width = 1-255, height =1-255, width = 0 - no wrapping performed, height = 0 - no clipping performed

Xpos – WORD

Description – pos relative to object origin if defined else relative to 0,0 otherwise

Valid Values: 0x0000-0xFFFF

Ypos – WORD

Description – pos relative to object origin if defined else relative to 0,0 otherwise

Valid Values: 0x0000-0xFFFF

NOTE: Colours in the range of 0x80F0 - 0x80FF are not valid colour indexes into VideoData LUTs since they only support up to 240 colours. Hence they must be interpreted as per the following table. These colours should be mapped into the specific device/OS system colours as best possible according to the table. In the standard Palm OS UI only 8 colours are used and some of these colours are similar to the other platforms but not identical, this is indicated with an asterix. The missing 8 colours will have to be set by the application.

Value	RGB Value	Name	WinCE	Palm OS	Epoc
0x80 F0	0x00 00 00	Black	0	255	0
0x80 F1	0x80 00 00	Dark Red	1		2
0x80 F2	0x00 80 00	Dark Green	2		3
0x80 F3	0x80 80 00	Dark Yellow	3		4
0x80 F4	0x00 00 80	Dark Blue	4	89*	5
0x80 F5	0x80 00 80	Dark Magenta	5		6
0x80 F6	0x00 80 80	Dark Cyan	6		7
0x80 F7	0x80 80 80	Medium Grey	248	50*	14
0x80 F8	0x FF 00 00	Red	249	125	8
0x80 F9	0x00 FF 00	Green	250	211*	9
0x80 FA	0xFF FF 00	Yellow	251	120	10
0x80 FB	0x00 00 FF	Blue	252		11
0x80 FC	0xFF 00 FF	Magenta	253		12
0x80 FD	0x00 FF FF	Cyan	254	90	13
0x80 FE	0xFF FF FF	White	255	0	15
0x80 FF	Transparent	-			-

GrafDefinition

This packet contains the basic animation parameters. The actual graphic object definitions are contained in the GrafData packets, and the animation control in the objControl packets.

Xpos - WORD

Description – XPos relative to object origin if defined relative to 0,0 otherwise

Valid Values:

Ypos - WORD

Description – XPos relative to object origin if defined relative to 0,0 otherwise

Valid Values:

FrameRate - WORD

Description – Frame delay in 8.8 fps

Valid Values:

FrameSize - WORD

Description – Frame size in twips (1/20 pel)– used for scaling to fit scene space

Valid Values:

FrameCount -WORD

Description – How many frames in this animation

Valid Values:

Time – DWORD

Description – Time stamp in 50ms resolution from start of scene

Valid Values:

VideoKey, VideoData, and AudioData

These packets contain codec specific compressed data. Buffer sizes should be determined from the information conveyed in the VideoDefn and AudioDefn packets. Other than the TypeTag VideoKey packets are identical to VideoData packets are serve identical purposes. The reason for the difference in type definition is to make keyframes visible at the file parsing level to facilitate browsing. VideoKey packets are an integral component of a sequence of VideoData packets, they are typically interspersed among them as part of the same packet sequence.

TextData

Textdata packets contain only the ASCII character codes for text to be rendered. Whatever Serif system font are available one the client device should be used to render these fonts. Serif fonts are to be used since proportional fonts require additional processing to render. In the case where the specified Serif system font style is not available, then the closest matching available font should be used.

Plain text is rendered directly without interpretation. White space characters other than LF (new line) characters and spaces are ignored. All text is clipped at scene boundaries.

The bounds box defines how text wrapping functions. The text is wrapped using the width, and clipped if it exceeds the height. If the bounds width is 0 then no wrapping occurs. If the height is 0 then no clipping occurs.

Table data is treated as Plain text with the exception of LF, which is used to denote end of rows and the CR character that is used to denote columns breaks.

WML and HTML is interpreted according to their respective standards, and the font style specified in this format is ignored. Images are not supported in WML and HTML.

To obtain streaming text data, new TextData packets are sent to update the relevant object. Also in normal text animation, the rendering of TextData can be defined using ObjectControl packets.

Forms are used to support interactive user input. The format and interpretation of this text data type is yet to be determined. Entering input to a form will generate UserControl packets and to be sent to the server.

GrafData

This packet contains all of the graphic shape and style definitions used for the graphics animation. This is a very simple animation data type. Each shape is defined by a path, some attributes and a drawing style. One graphic object may be composed of an array of paths in any one GraphData packet. Animation of this graphic object can occur by clearing or replacing individual shape records array entires in the next frame, adding new records to the array can also be performed using the CLEAR and SKIP path types.

GraphData Packet

Description	Type	Comment
NumShapes	BYTE	Number of shape records to follow
Primitives	SHAPERecord[]	Array of Shape Definitions

ShapeRecord

Description	Type	Comment
Path	BYTE	Sets the path of the shape + DELETE operation
Style	BYTE	Defines how path is interpreted and rendered
Offset	IPOINT	
Vertices	DPOINT[]	Length of array given in Path low nibble
FillColour	WORD[]	Number of entries depend on fill style and # vertices
LineColour	WORD	Optional field determined by style field

Path – BYTE

Description – Sets the path of the shape in the high nibble and the # vertices in low nibble

Low 4 Bits Value: 0 – 15: number of vertices in poly paths

High 4 Bits Value: ENUMERATED: 0 – 15 defines the path shape

Value	Path	Comment
0	CLEAR	Deletes SHAPERECORD definition from array
1	SKIP	Skips this SHAPERECORD in the array
2	RECT	Description – topleft corner, bottom right corner Valid Values: (0..4096, 0..4096), [0..255, 0..255]…
3	POLY	Description – # points, initial xy value, array of relative pt coords Valid Values: 0..255, (0..4096, 0..4096), [0..255, 0..255]…
4	ELLIPSE	Description – centre coord, major axis radius, minor axis radius Valid Values: (0..4096, 0..4096), 0..255, 0..255
5 – 15		RESERVED

Style – BYTE

Description – Defines how path is interpreted

Low 4 Bits Value: 0 – 15 line thickness

High 4 Bits: BITFIELD: path rendering parameters. The default is not draw the shape at all so that it operates as an invisible hot region.

Bit [4]: CLOSED -	If bit set then path is closed
Bit [5]: FILLFLAT -	Default is no fill – if both fills then do nothing
Bit [6]: FILLSHADE -	Default is no fill – if both fills then do nothing
Bit [7]: LINECOLOR -	Default is no outline

UserControl

These are used to control the user-system and user-object interaction events. They are used as a back channel to return user interaction back to a server to effect server side control. However if the file is not being streamed these user interactions must be handled locally by the client. A number of actions can be defined for user-object control in each packet. The following actions are defined in this version. The user-object interactions need not be specified except to notify the server that one has occurred since the server knows what actions are valid.

User-system interactions	User-Object interactions
Pen events (up, down, move, dblclick)	Set 2D position, visibility (self, other)
Keyboard events	Play / Pause system control
Play control (play, pause, frame advance, stop)	Hyperlink - Goto # (Scene, frame, label, URL)
Return Form Data	Hyperlink - Goto next/prev, (scene, frame)

	Hyperlink - Replace object (self, other)
	Hyperlink – Server Defined

The user-object interaction depends on what actions are defined for each object when they are clicked on by the user. The player may know these actions through the medium of ObjectControl messages. If it does not then they must be forwarded to an online server for processing. With user-object interaction the identification of the relevant object is indicated in the BaseHeader obj_id field. This applies to OBJCTRL and FORMDATA event types. For user-system interaction the value of the obj_id field must equal 255. The Event type in UserControl packets specifies the interpretation of the key, HiWord and LoWord data fields.

Event – BYTE

Description – User Event Type

Valid Values: enumerated 0-255

Value	Event Type	Comment
0	PENDOWN	User has put pen down on touch screen
1	PENUP	User has lifted pen up from touch screen
2	PENMOVE	User is dragging pen across touch screen
3	PENDBLCLK	User has double clicked touch screen with pen
4	KEYDOWN	User has pressed a key
5	KEYUP	User has pressed a key

6	PLAYCTRL	User has activated a play/pause/stop control button
7	OBJCTRL	User has clicked/activated an AV object
8	FORMDATA	User is returning form data
9-255	-	RESERVED

key, HiWord and LoWord – BYTE, WORD, WORD

Description – parameter data for different event types

Valid Values: The interpretation of these fields is as follows

Event	Key	HiWord	LoWord
PENDOWN	Key code if key held down	X position	Y position
PENUP	Key code if key held down	X position	Y position
PENMOVE	Key code if key held down	X position	Y position
PENDBLCLK	Key code if key held down	X position	Y position
KEYDOWN	Key code	Unicode Key code	2nd key held down
KEYUP	Key code	Unicode Key code	2nd key held down
PLAYCTRL	Stop=0, Start=1, pause = 2	RESERVED	RESERVED
OBJCTRL	Pen Event ID	Keycode if key held	RESERVED

		down	
FORMDATA	RESERVED	Length of data field	RESERVED

Time – WORD
Description – Time of user event = sequence number of activated object
Valid Values: 0-0xFFFF

Data – (RESERVED - OPTIONAL)
Description – Text strings from form object
Valid Values: 0...65535 bytes in length

Note: In the case of the PLAYCTRL events that pausing repeatedly when play is already paused should invoke a frame advance response from the server. Stopping should reset play to the start of the file/stream.

ObjectControl

ObjectControl packets are used to define the object-scene and system-scene interaction. They also specifically define how objects are rendered and how scenes are played out. A new OBJCTRL packet is required for each frame to coordinate individual object layout. A number of actions can be defined for an object in each packet. The following actions are defined in this version

Object-system actions	System-scene actions
Set 2D/3D position	Goto # (Scene, frame, label, URL)

Set 3D Rotation	Goto next, previous, (scene, frame)
Set scale/size factor	Play / Pause
Set visibility	Mute audio
Set label/title (for use as in tool tips)	IF (scene, frame, object) THEN DO (action)
Set background colour (nil = transparent)	
Set tweening value (for animations)	
Begin /end / duration / repeat (loop) *implicit*	

ActionMask [OBJECT scope] – WORD
~~Description – Bit field - This defines what actions are specified in this record and the~~ parameters to follow. There are two versions of this one for object the other for system scope.
Valid Values: For objects each one of the 16 bits in the ActionMask identifies an action to be taken. If a bit is set then additional associated parameter values must follow this field.

Bit: [15] CONDITION	– What is needed to perform these actions
Bit: [14] MOVETO	– screen position
Bit: [13] ZORDER	– Depth
Bit: [12] ROTATE	– 3D Orientation
Bit: [11] ALPHA	– Transparency
Bit: [10] SCALE	– Scale / size
Bit: [9] VOLUME	– loundness
Bit: [8] BACKFILL	– colour of object background

Bit: [7] TWEEN – tweening amount

Bit: [6] PROTECT – limit user modification of scene objects

Bit: [5] BEHAVIOR – what to do on mouse/pen clicks

Bit: [4] LOOP – repeat the next # actions (if set else BREAKLOOP)

Bit: [3] LABEL – string for tool tip like function

Bit: [2] HLINK – Sets hyperlink target

Bit: [1] – RESERVED

Bit: [0] Extension – Extended Action Mask (ACTIONEXTEND) follows

ActionExtend [OBJECT scope] – WORD

Description – Bit field - RESERVED

ActionMask [SYSTEM scope] – WORD

Description – Bit field - This defines what actions are specified in this record and the parameters to follow. There are two versions of this one for object the other for system scope.

Valid Values: For systems each one of the 16 bits in the ActionMask identifies an action to be taken. If a bit is set then additional associated parameter values must follow this field

Bit: [15] CONDITION – What is needed to perform these actions

Bit: [14] PLAYSTATE – if playing pause indefinitively

Bit: [13] SNDMUTE – if sounding then mute if muted then sound

Bit: [12] HLINK – jump to hyperlink (URL)

Bit: [11] GOTO – next/prev scene

Bit: [10] NAME – BYTE[] string for referencing

Bit: [9] BACKFILL – WORD change the SceneDefinition BackFill colour

Bit: [8] PROTECT – limit user modification of scene object

Bit: [7] SETFLAG- – Sets user assignable flag value

Bits: [6-0] - RESERVED

Params – BYTE array

Description – Byte array. Most of the actions defined in the above bit fields require additional parameters. The required parameters as indicated by the bit field value being set are specified here in the same order as the bit field from top (15) to bottom (0). These parameters may include optional fields, these are shown as yellow rows in the tables below.

CONDITION bit – Consists of one or more state records chained together, each record can also have an optional frame number field after it. The conditions within each record are logically ANDed together. For greater flexibility additional records can be chained through bit 0 to create logical OR conditions. In addition to this, multiple, distinct definition records may exist for any one object creating multiple conditional control paths for each object.

Param	Type	Comment
State	DWORD	What is needed to perform these actions, bit-field (logically ANDed) Bit: [31] playing // continuous playing Bit: [30] paused // playing is paused Bit: [29] stream // streaming from remote server Bit: [28] stored // playing from local storage Bit: [27] buffered // is object frame # buffered? (true if stored) Bit: [26] overlap // what object do we need to be dropped on? Bit: [25] event // what user event needs to be happening Bit: [24] wait // do we wait for conditions to become true

		Bit: [23-17] RESERVED Bit: [1..16] userflags // 16 user assignable system flags Bit: [0] OrState // OrState condition record follow
Frame	WORD	(optional) frame number for bit 27 condition
Object	BYTE	(optional) object ID for bit 26 condition, invisible objects can be used
Event	WORD	High BYTE: the event field from the UserControl Packet Low BYTE: the key field from the UserControl Packet
State	DWORD	Same bit field as the previous state field, but is logically ORed with it
...	DWORD	...

MOVETO bit set

Param	Type	Comment
Xpos	WORD	X position to move to, relative to current pos
Ypos	WORD	Y position to move to, relative to current pos

ZORDER bit set

Param	Type	Comment
Depth	WORD	Depth increases away from viewer, values of 0,256,512,768 etc reserved

ROTATE bit set

Param	Type	Comment
Xrot	BYTE	X axis rotation, absolute in degrees * 255 / 360,

Yrot	BYTE	Y axis rotation, absolute in degrees * 255 / 360
Zrot	BYTE	Z axis rotation, absolute in degrees * 255 / 360

ALPHA bit set

Param	Type	Comment
alpha	BYTE	Transparency 0 = transparent, 255 = fully opaque

SCALE bit set

Param	Type	Comment
scale	WORD	Size / Scale in 8.8 fixed int format

VOLUME bit set

Param	Type	Comment
vol	BYTE	Sound volume 0 = softest, 255 = loudest

BACKFILL bit set

Param	Type	Comment
fillcolr	WORD	Same format as SceneDefinition Backcolor (nil = transparent)

TWEEN bit set

Param	Type	Comment
tween	BYTE	tweening amount for animation/morphing

PROTECT bit set

Param	Type	Comment
Protect	BYTE	limit user modification of scene objects bit field, bit set = disabled

		Bit: [7] move // prohibit dragging objects Bit: [6] alpha // prohibit changing alpha value Bit: [5] depth // prohibit changing depth value Bit: [4] clicks // disable click through behaviour Bit: [3..0] // RESERVED

BEHAVIOR bit set – if behaviour bit 4 set then a HLINK field must be defined

Param	Type	Comment
Behave	BYTE	what to do on mouse/pen clicks (link for self/ other) or animate self/other: setflag, jump to link, move, delete etc Bit: [7] OTHER // if not self apply to another object # below Bit: [6] JUMPTO // replace target object with another Bit: [5] SETFLAG // set / reset user flag value Bit: [4] ENDLOOP // if looping control then break it ~~Bit: [3] BUTTON // define button animation~~ Bit: [2] RESUME // resume playing after general pause Bit: [1..0] // RESERVED
Target	BYTE	OTHER target object / scene number for this action
Jump	WORD	high BYTE - what to replace, (file, scene, object) low BYTE - replacement object / scene ID number (255 = HLINK URL)
Uflag	BYTE	same as SETFLAG field
Button	BYTE	What is the pressed button object image object

LOOP bit set

Param	Type	Comment
Repeat	BYTE	Repeat the next # actions for this object – clicking on object

		to break loop

LABEL bit set

Param	Type	Comment
Label	BYTE	String for tool tip like function / referencing

SETFLAG bit set

Param	Type	Comment
Flag	BYTE	Bottom 4 bits = flag number, top 4 bits if true set flag else reset flag,

HLINK bit set

Param	Type	Comment
hLink	BYTE []	Sets hyperlink target URL for click through

GOTO bit set

Param	Type	Comment
scene	WORD	Goto scene #

Many modifications will be apparent to those skilled in the art without departing from the spirit and scope of the present invention as hereinbefore described.

DATED this 7th day of July 2000

ActiveSky, Inc.
By its Patent Attorneys
DAVIES COLLISON CAVE

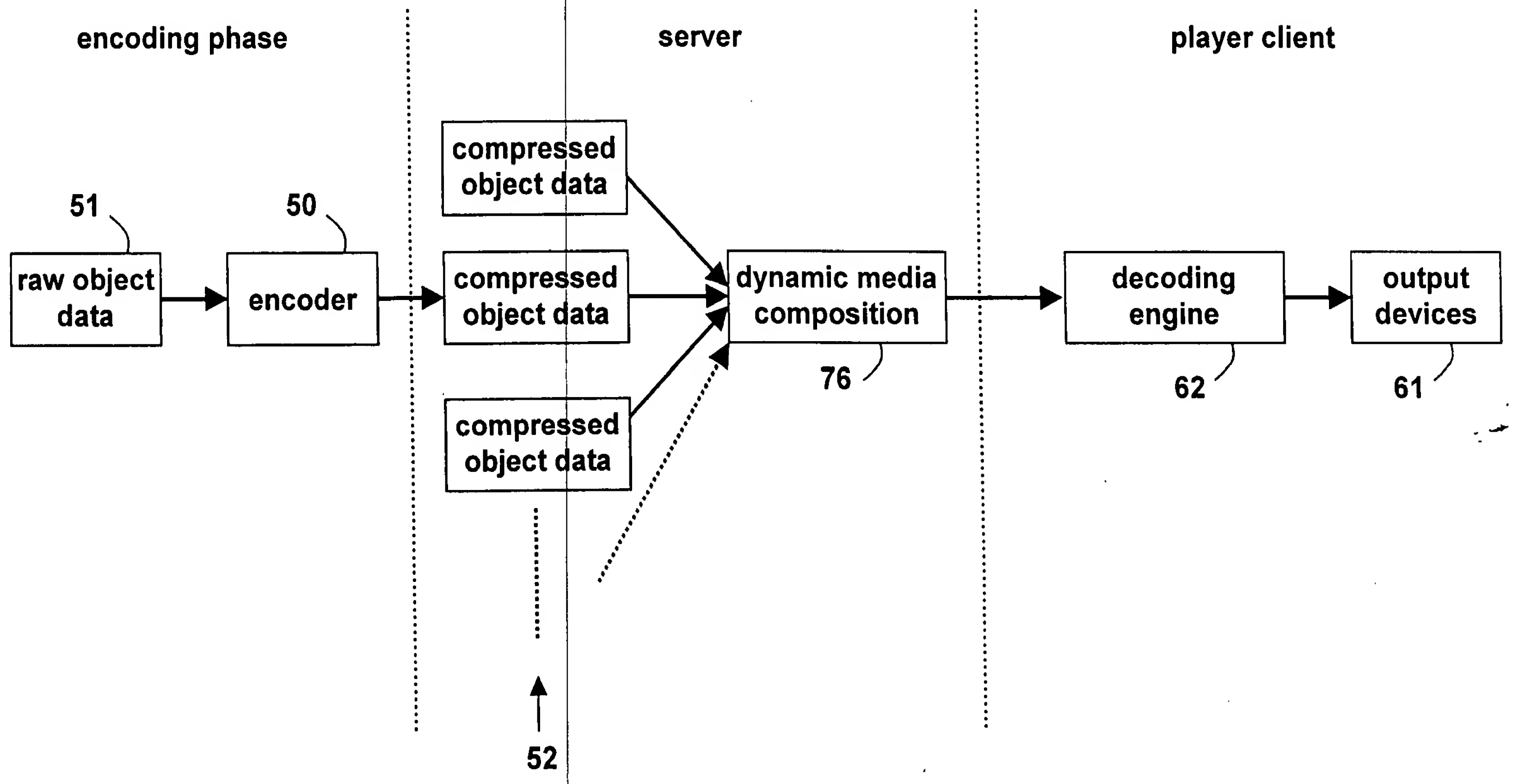
encoding phase
server
player client
51
raw object data
50
encoder
compressed object data
compressed object data
compressed object data
52
dynamic media composition
76
decoding engine
62
output devices
61

Figure 1

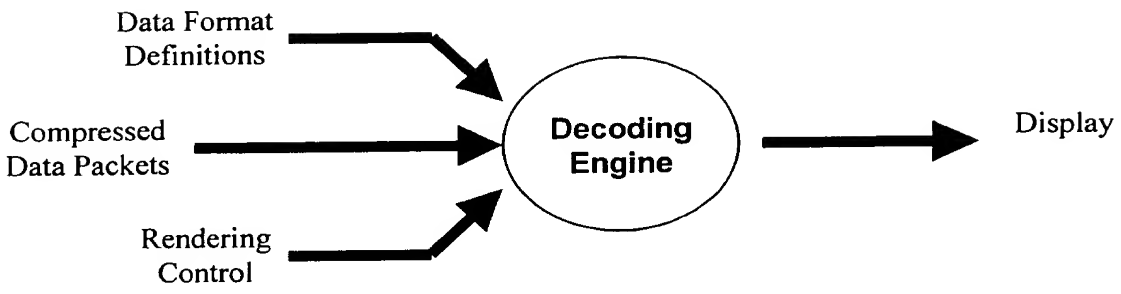

Figure 2

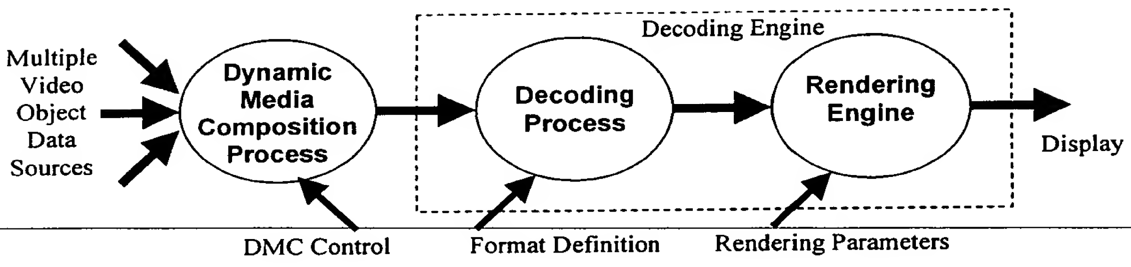

Figure 3

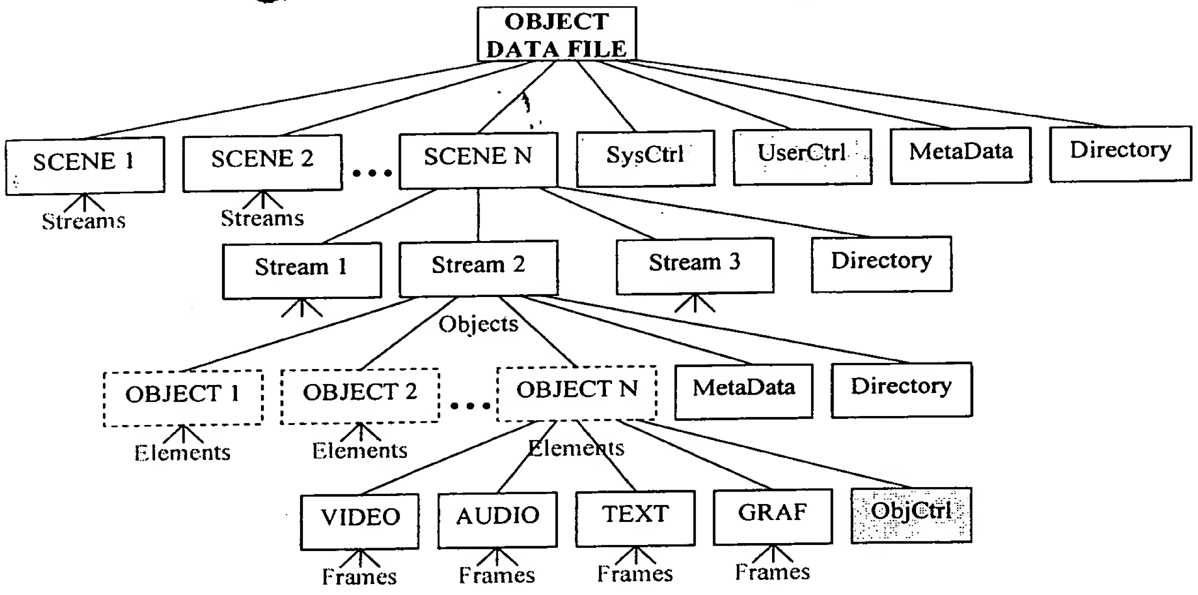

Figure 4

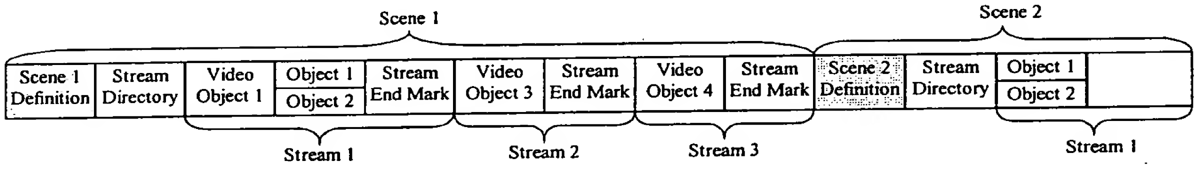

Figure 5

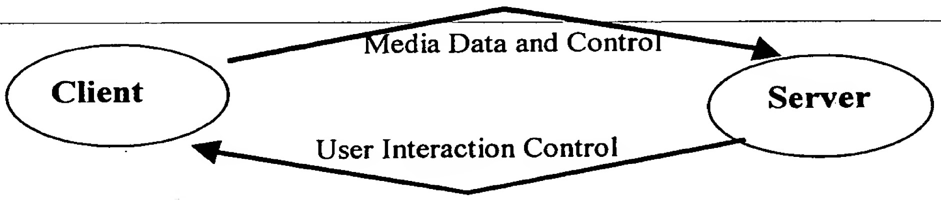

Figure 6

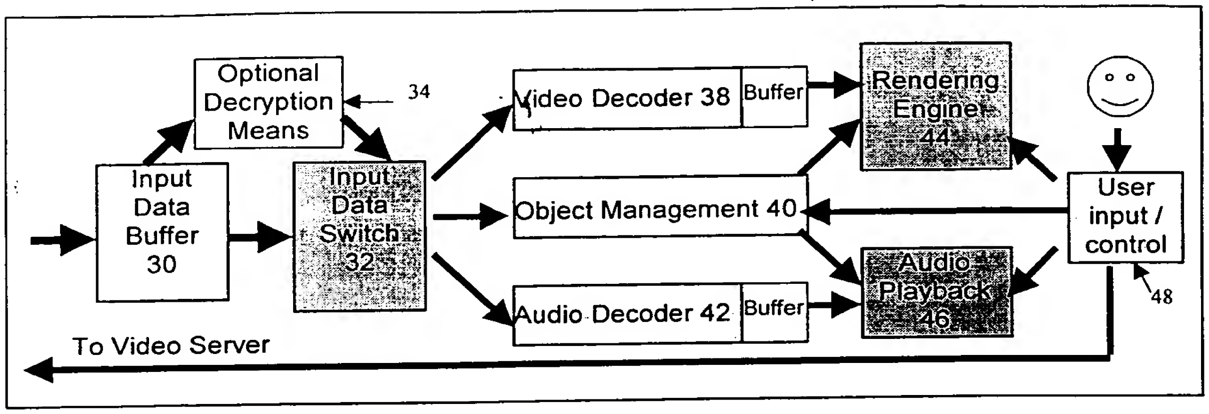

Figure 7

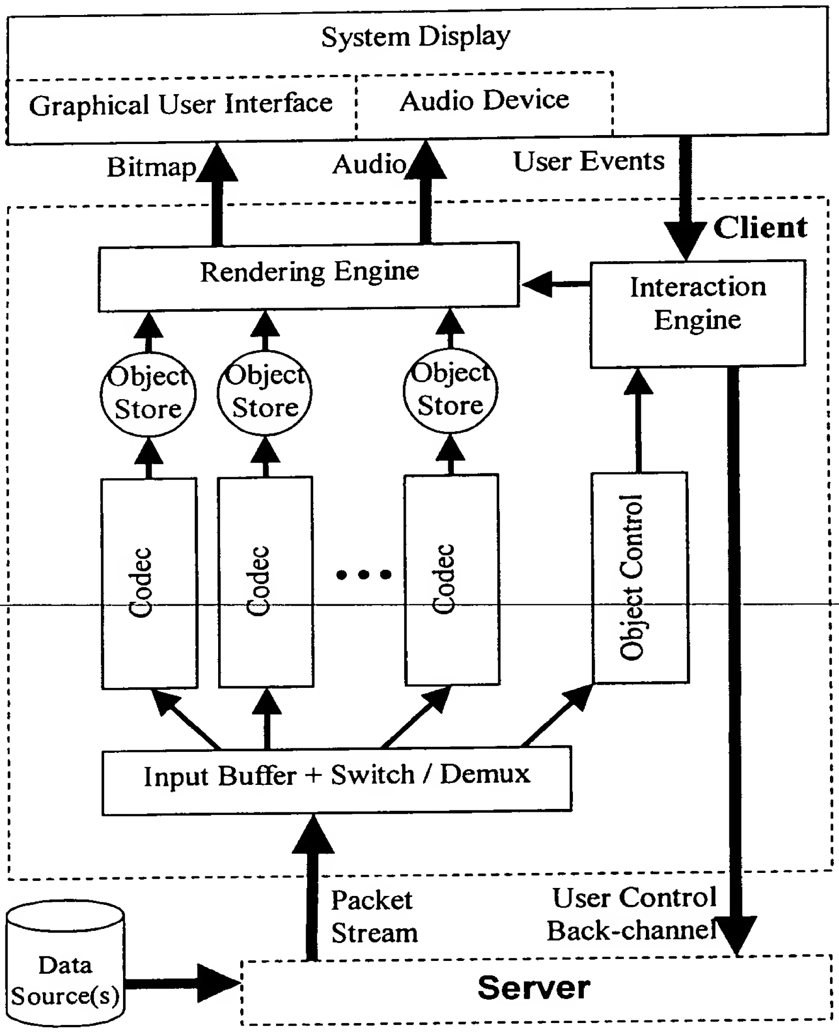

Figure 8

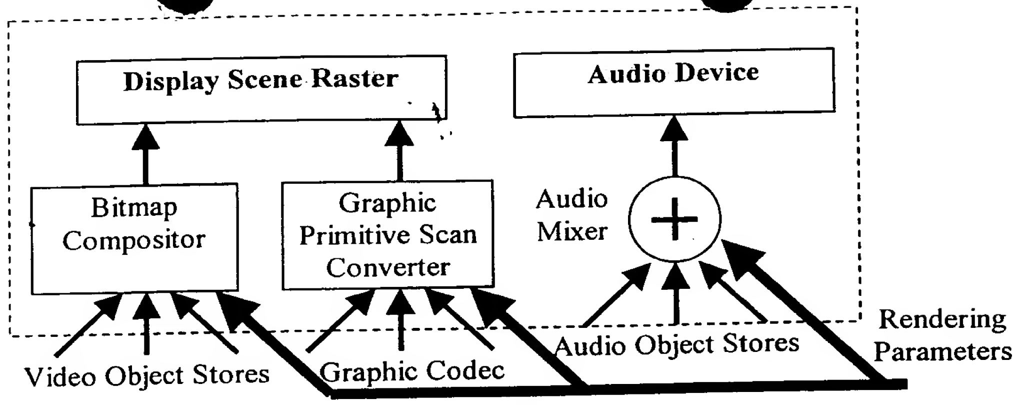

Figure 9

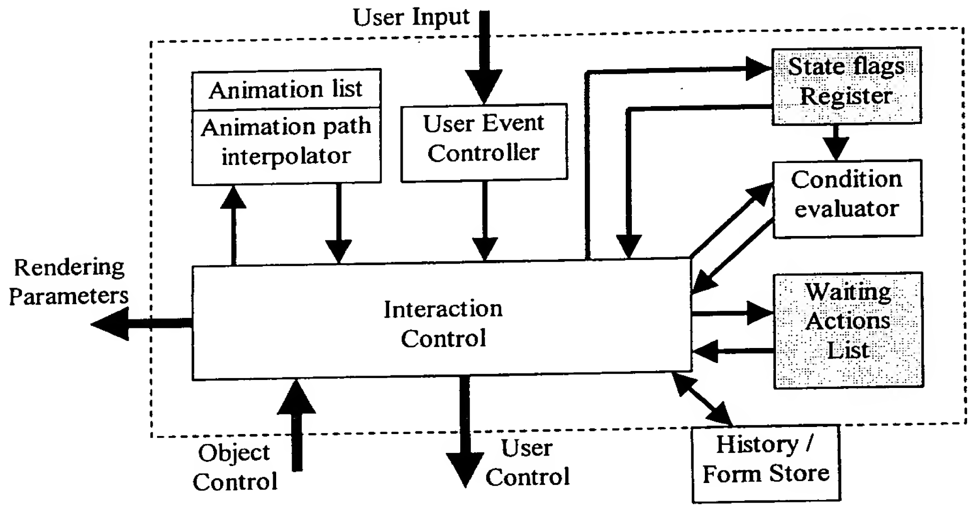

Figure 10

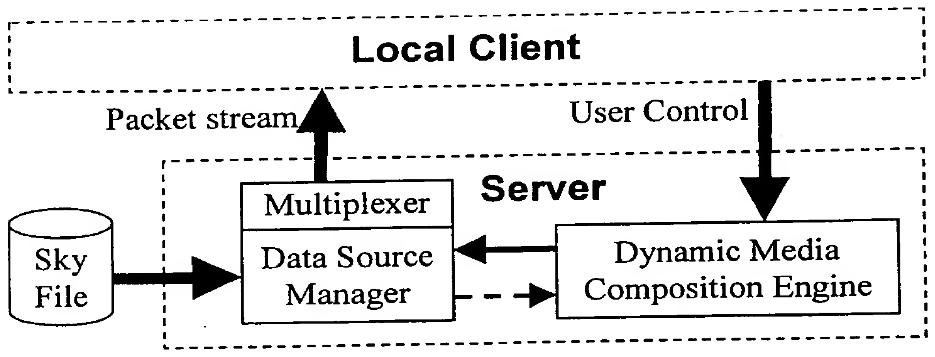

Figure 11

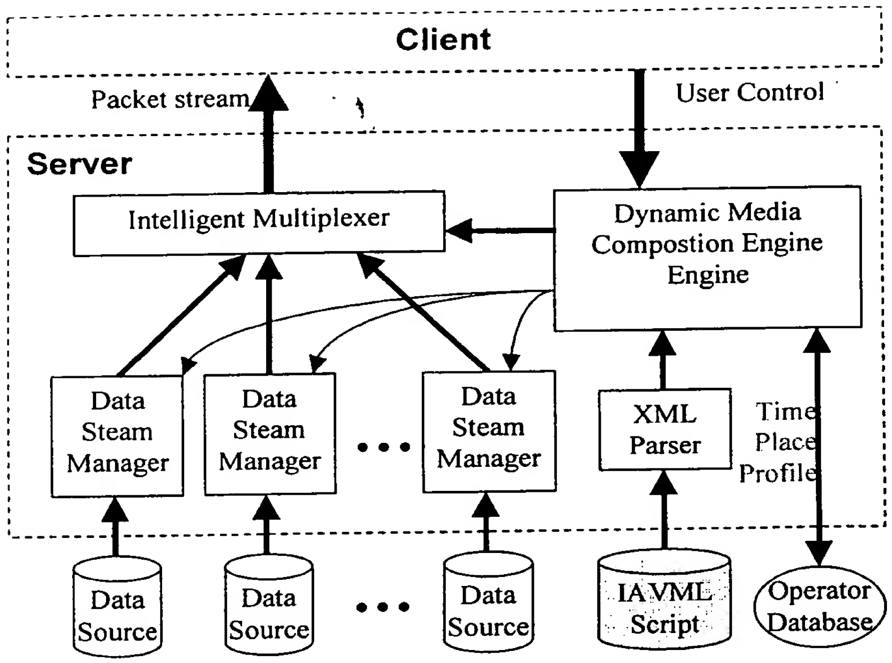

Figure 12

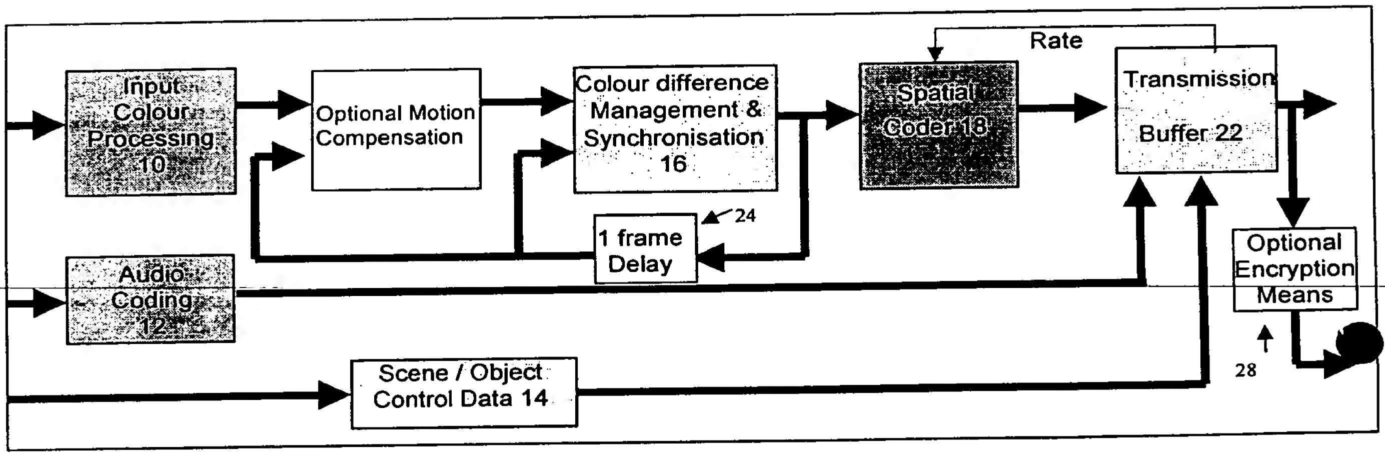
Input Colour Processing 10
Optional Motion Compensation
Colour difference Management & Synchronisation 16
Spatial Coder 18
Rate
Transmission Buffer 22
1 frame Delay
24
Audio Coding 12
Optional Encryption Means
28
Scene / Object Control Data 14

Figure 13

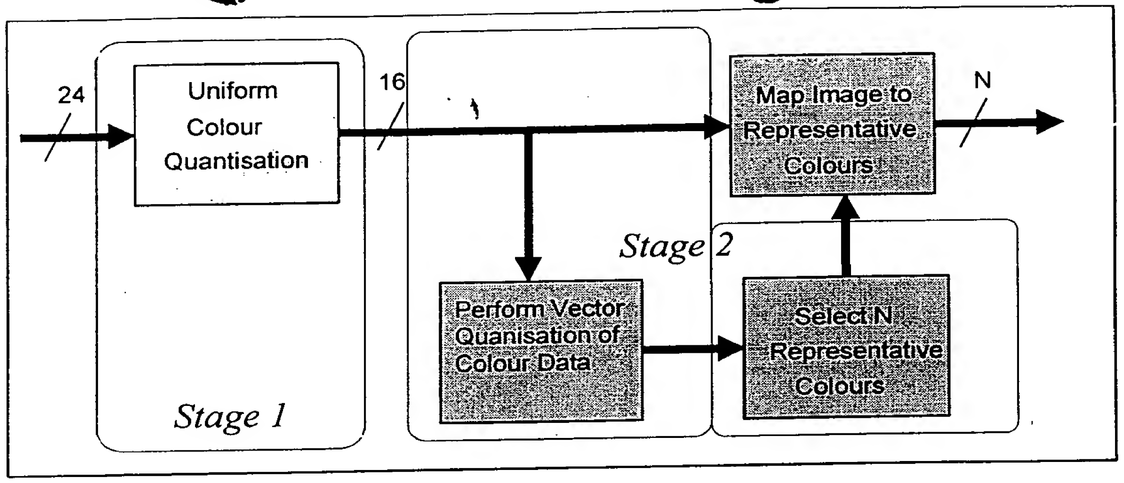
24
Uniform Colour Quantisation
16
Map Image to Representative Colours
N
Stage 2
Perform Vector Quanisation of Colour Data
Select N Representative Colours
Stage 1

Figure 14

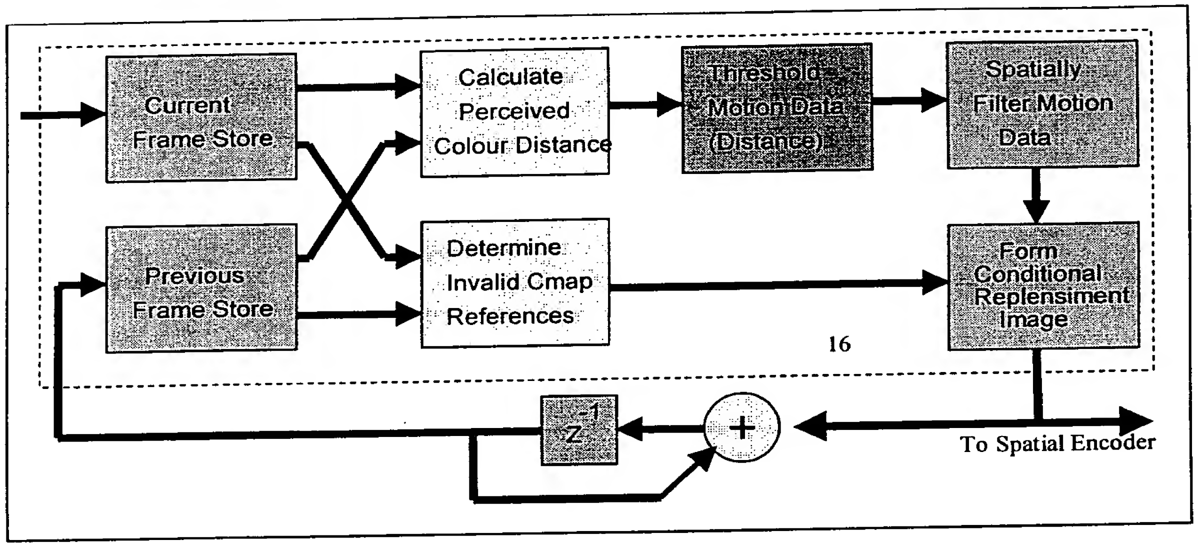
Current Frame Store
Previous Frame Store
Calculate Perceived Colour Distance
Determine Invalid Cmap References
Threshold Motion Data (Distance)
Spatially Filter Motion Data
Form Conditional Replensiment Image
16
Z-1
+
To Spatial Encoder

Figure 15

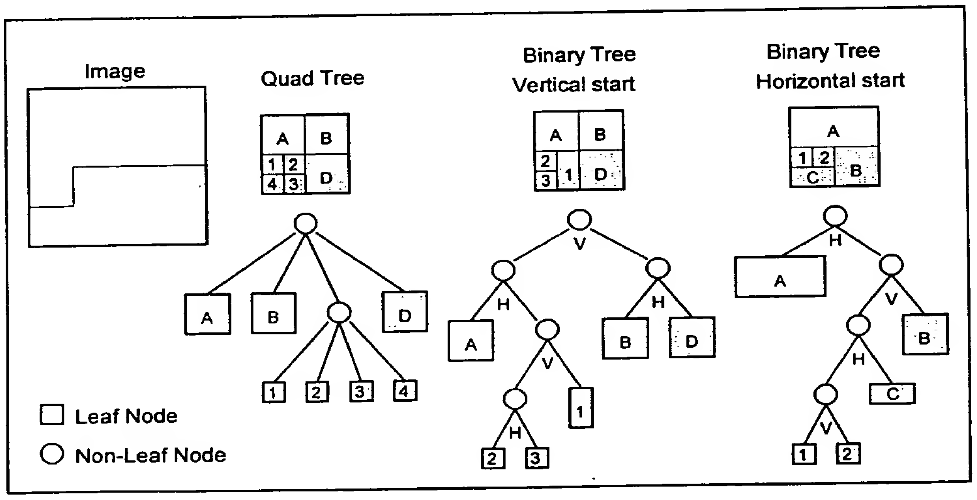

Figure 16

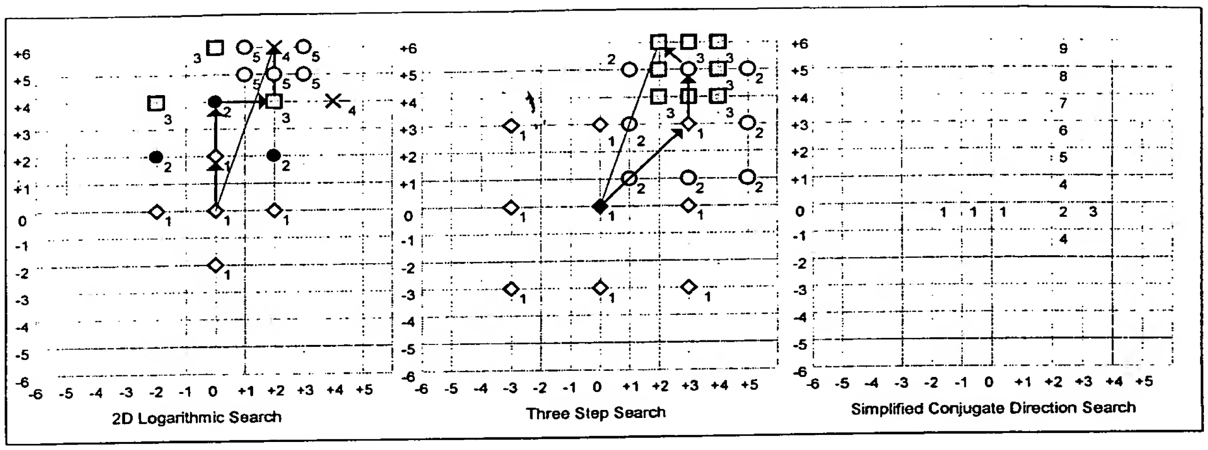

Figure 17

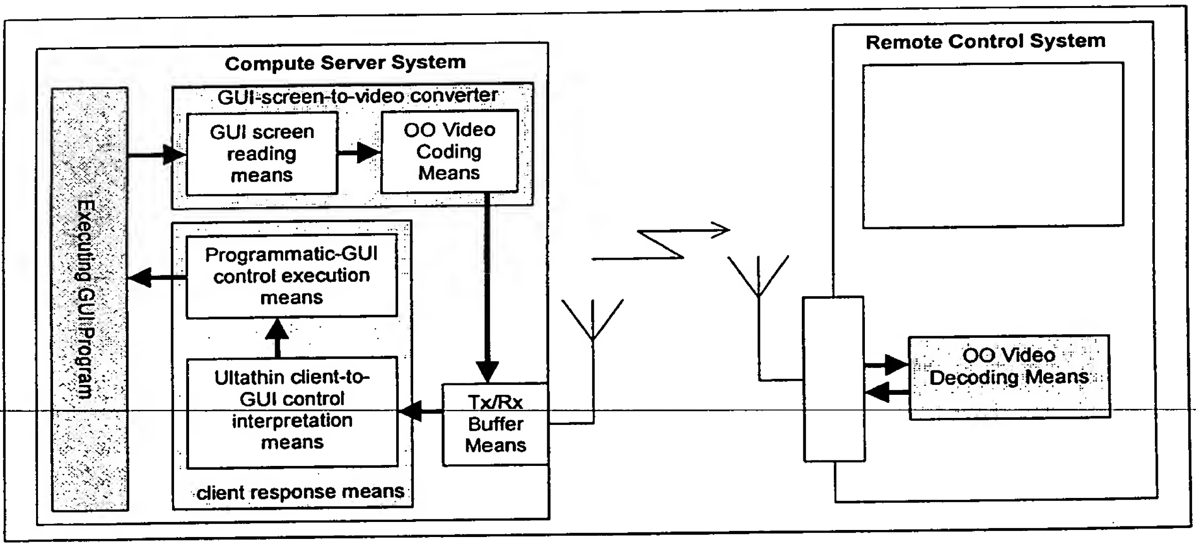

Figure 18

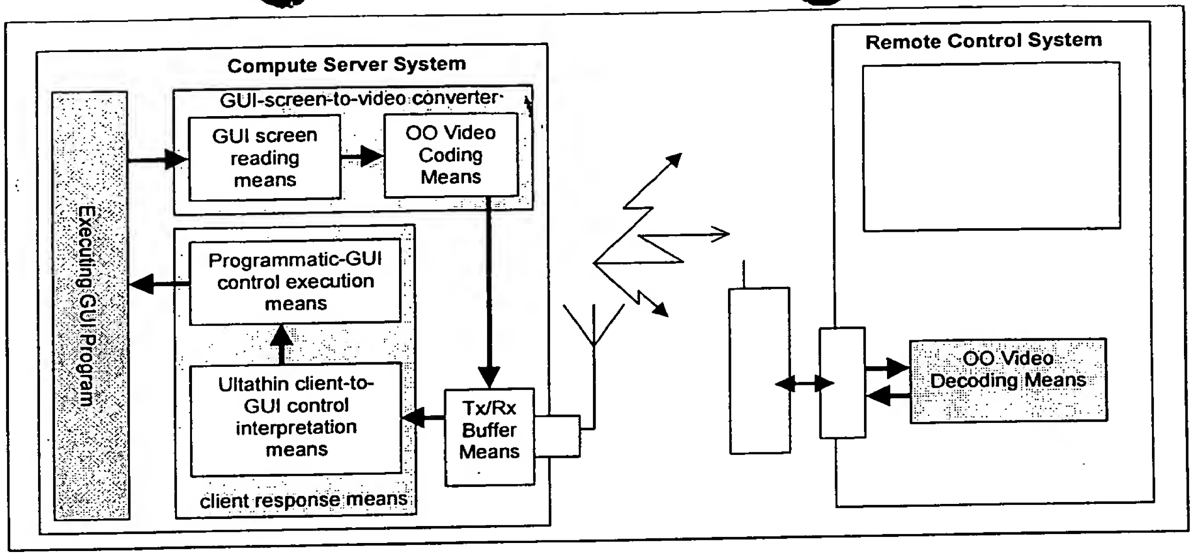

Figure 19

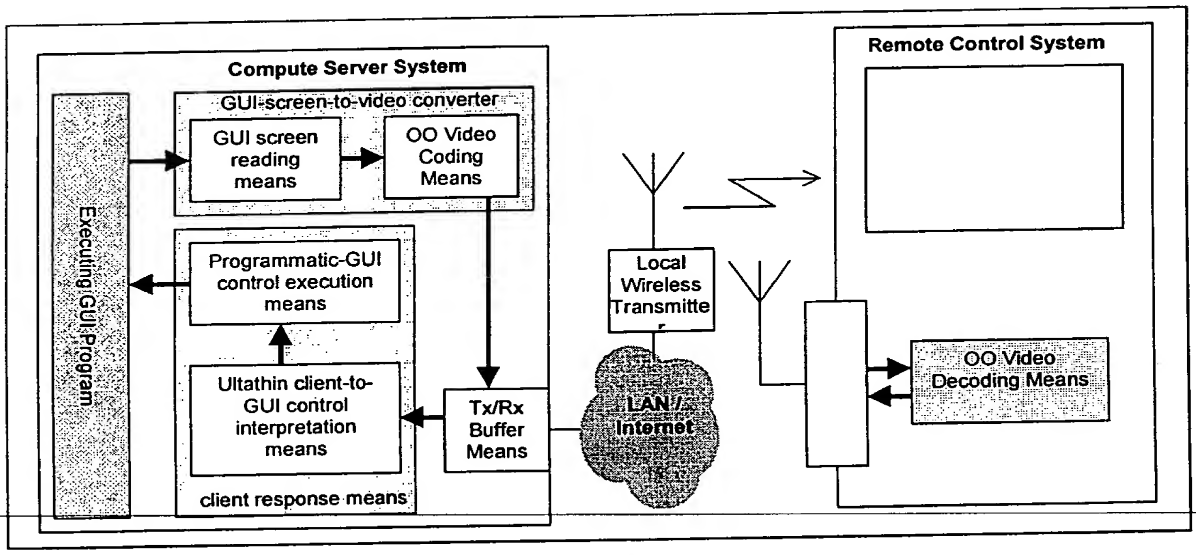

Figure 20

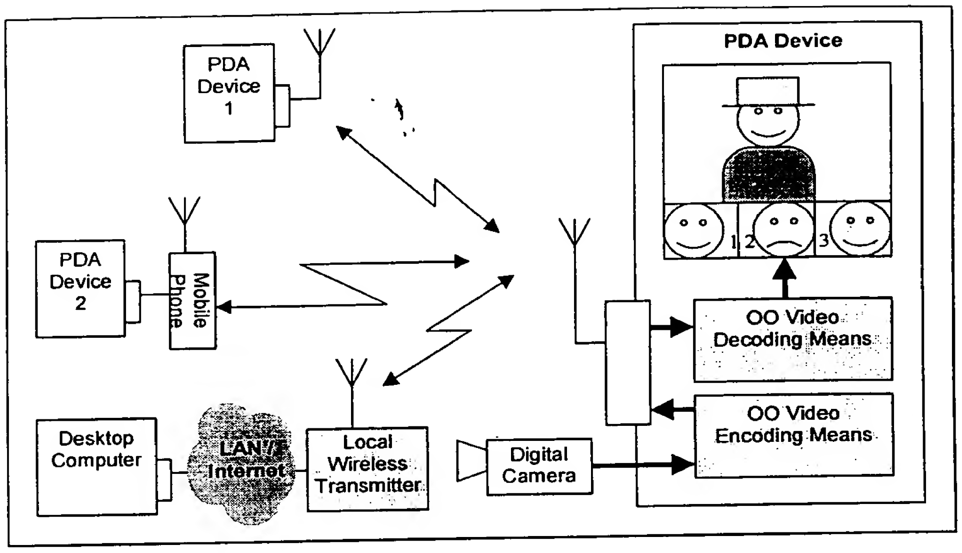

Figure 21

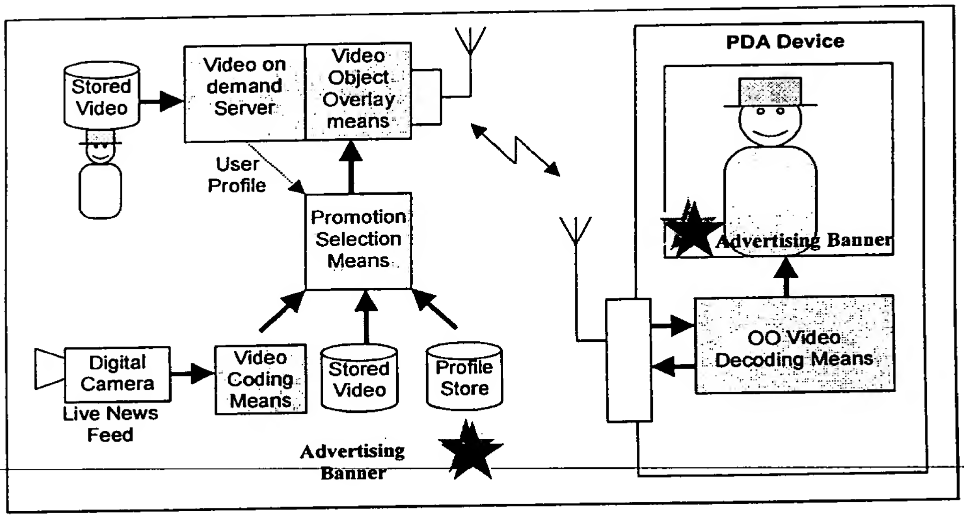

Figure 22

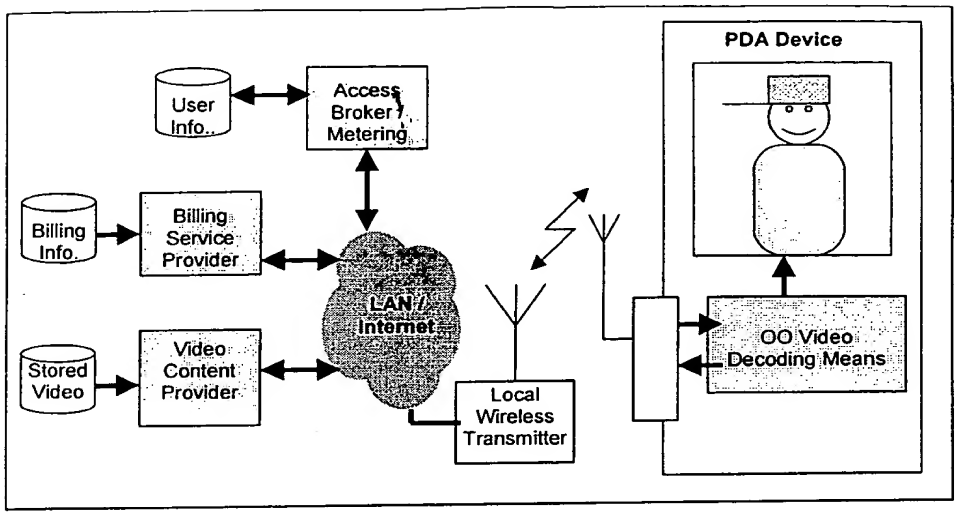

Figure 23

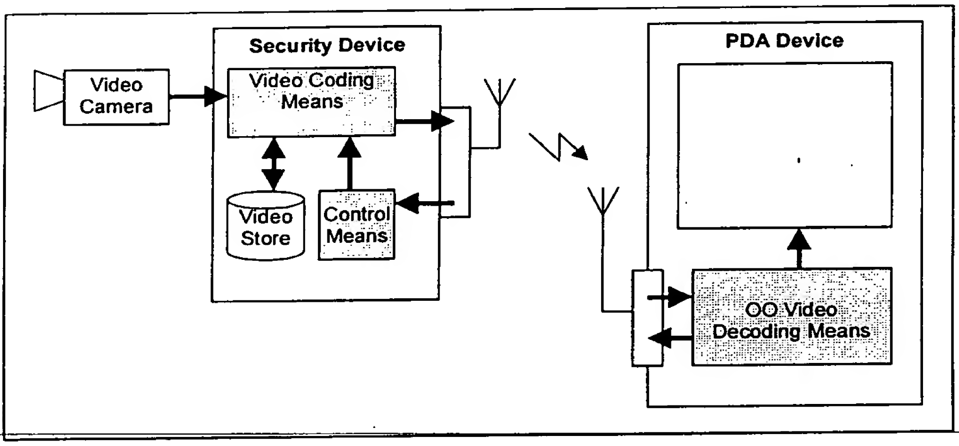

Figure 24

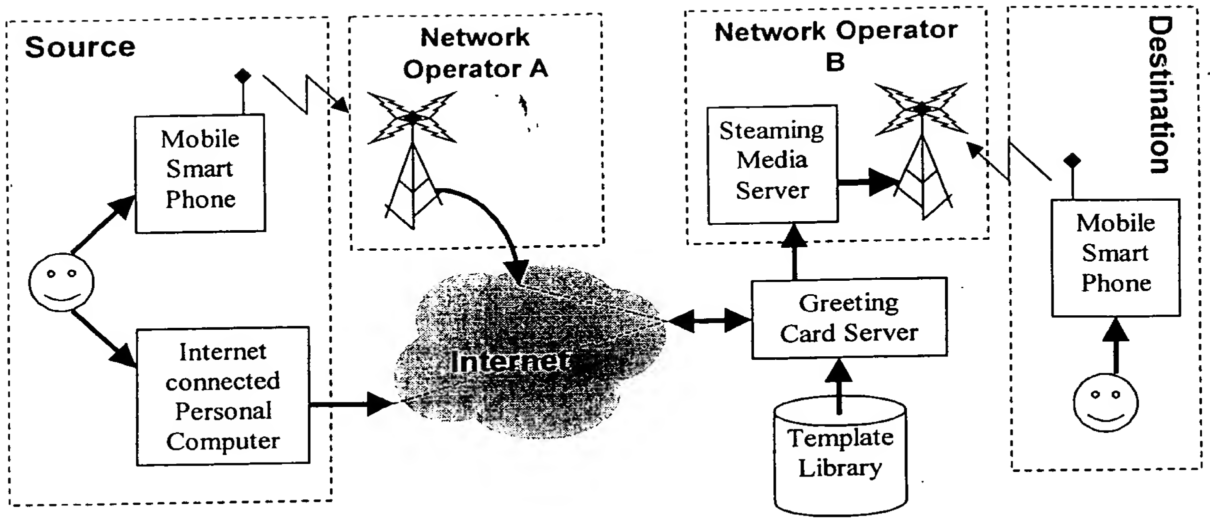
Source
Mobile Smart Phone
Internet connected Personal Computer
Network Operator A
Internet
Network Operator B
Steaming Media Server
Greeting Card Server
Template Library
Destination
Mobile Smart Phone

Figure 25